July 1991

Report No. STAN-CS-92-1405

Thesis

PB96-149588

Kaleidoscope: A Model-Based Grammar-Driven Menu Interface for Databases

by

Sang Kyun Cha

Department of Computer Science

Stanford University

Stanford, California 94305

KALEIDOSCOPE: A MODEL-BASED GRAMMAR-DRIVEN MENU INTERFACE FOR DATABASES

A DISSERTATION
SUBMITTED TO THE DEPARTMENT OF ELECTRICAL ENGINEERING
AND THE COMMITTEE ON GRADUATE STUDIES
OF STANFORD UNIVERSITY
IN PARTIAL FULFILLMENT OF THE REQUIREMENTS
FOR THE DEGREE OF
DOCTOR OF PHILOSOPHY

By
Sang Kyun Cha
July 1991

I certify that I have read this dissertation and that in my opinion it is fully adequate, in scope and in quality, as a dissertation for the degree of Doctor of Philosophy.

[illegible]

Gio Wiederhold
(Principal Advisor)

I certify that I have read this dissertation and that in my opinion it is fully adequate, in scope and in quality, as a dissertation for the degree of Doctor of Philosophy.

[illegible]

Terry Winograd

I certify that I have read this dissertation and that in my opinion it is fully adequate, in scope and in quality, as a dissertation for the degree of Doctor of Philosophy.

John T. Gill III

John T. Gill III

Approved for the University Committee on Graduate Studies:

Dean of Graduate Studies

Abstract

Most database interfaces provide poor guidance on ad hoc query formulation, forcing end users to learn, and to recall precisely the query language and the database. The research on the so-called natural language interface (NLI) promised a solution to this problem. However, in practice, with the difficulty of developing a large body of machine-interpretable knowledge on human linguistic behavior, the success of NLI systems is inevitably limited. NLI users are still required to learn and recall the limitations of a specific system.

This thesis presents the approach of Kaleidoscope, a cooperative query interface for relieving the user's burden of learning and recalling. Kaleidoscope provides the user with an English-like query language (EnQL) for interaction with database systems. It guides the user's query formulation actively via a sequence of menu interactions. Based on a grammar specifying the syntax and semantics of EnQL, the interface proposes legitimate query constituents step by step as menu choices. The objective of this grammar-driven menu guidance is to enable users to construct a meaningful query by recognizing choices that match their mental query. The interface provides additional intraquery conceptual guidance to ensure the integrity of a partial query.

The central thesis of this work is that a data model plays a crucial role in the Kaleidoscope's style of interfaces, as a query language conveys the underlying conceptualization of data to the user. The design of grammar, lexicon, and query translator follows a formally defined data model. The absence of an explicit model leads to the ad hoc design of these components, harming the system's transportability. In the model-based approach, grammar design focuses on unambiguously realizing references to model concepts. As a result, all user queries are meaningful with respect to the underlying data model. The model also provides a basis of defining two other domain-independent modules: a query translator and a set of procedures for automatically generating lexicon entries from the schema.

The major technical contribution of this thesis is a data model formalizing the conceptual structure of restricted English queries. Existing data models are inadequate for near-natural language interaction with database systems because of a significant conceptual gap between common English concepts and database representation of such concepts. EnQL, based on our model, enables the user to express significantly more concise queries than SQL, often by an order of magnitude. To provide a complete normative design framework, this thesis also presents a cost model of user query production when using grammar-driven menu interfaces. This model is useful for evaluating alternative interface designs.

Acknowledgements

I am greatly indebted to my advisor Gio Wiederhold for his generous support through my years at Stanford. He directed me into this interdisciplinary area of research with his professional insight and confidence, and guided me with invaluable advice until finishing it. After all, he was not just a research supervisor but a great teacher who set the cornerstones of my problem-solving mind.

I would also like to thank other members of my reading committee. I am fortunate to have Terry Winograd as one of them. In one of my early Stanford quarters, he enlightened me on the phenomenological foundation of language, cognition, and computation through his course. As a devoted reader, he provided me with detailed and objective feedback on this research. John T. Gill is the other reader who devoted his time to serving on my committee. His interest in my research was an encouragement. Teresa Meng in her office during the Christmas break kindly agreed to chair the committee.

Charles Kellogg at Lockheed Artificial Intelligence Center is an unofficial reader of this thesis. His extensive feedback on the early draft helped me in improving the quality of presentation. Craig Thompson was my mentor during my internship at Texas Instruments, Inc. This work has undoubtedly benefited by his generous support of my NLMenu exercise over TI's VLSI design database. I am grateful to Jack Milton for his careful reading of a part of this thesis presented elsewhere. He also devoted his precious time during the Christmas break to scheduling my thesis defense as the first CS545 seminar of the year 1991. I wish to thank my colleagues at IBM Palo Alto Scientific Center for tolerating the first dry run of my oral presentation and providing me with constructive feedback. This work has grown out of the common framework of the Stanford KBMS research. I would like to thank all of my past and current colleagues.

A number of friends made my Stanford life enjoyable and fruitful beyond completing the Ph.D. requirements. Voy Wiederhold has acted a caring mother at Stanford. Keith Hall and Toshi Matsushima are buddies who accompanied me to the city for Korean B.B.Q. after finishing my thesis defense. Surajit Chaudhuri is an outspoken reminder of an overdue homework in life. B.S. Lee was an unsuccessful preacher sympathetic with me. Peter Rathmann has been an excellent coordinator of KBMS activities.

Finally, I wish to extend my thanks to my family and professors in Korea for their moral support. Their presence has been the light guiding me to the end of a long and winding tunnel.

This research was supported by the DARPA Contract N00039-84-C-0211 for Knowledge Based Management Systems. The overseas graduate study fellowship of the Korean Ministry of Education helped me to start my computer science education at Stanford.

Contents

List of Tables

List of Figures

Chapter 1

Introduction

Scientific education makes use of no equivalent for the art museum or the library of classics, and the result is a sometimes drastic distortion in the scientist's perception of his discipline's past ... Inevitably those remarks will suggest that the members of a mature scientific community is, like the typical character of Owell's *1984*, the victim of a history rewritten by the powers that be.

– Thomas S. Kuhn, *The Structure of Scientific Revolutions* (1970)

The recent advances in hardware and network technology enable large organizations to move to a distributed model of computation from the traditional mainframe-based model. This distributed model increases the end-user availability and the local autonomy of operation. As the database management technology also matures, the distributed environment stimulates great interest in the end user's creation and maintenance of one's own database and encourages its sharing with others. With a limited number of database specialists, the user interface for supporting the end user's autonomous data access and management activities is critical for materializing the user's interest.

While task-oriented window-based interfaces best serve routine data access and data management functions, it is desirable to provide database interfaces with ad hoc querying power. Task-oriented interfaces do not meet this requirement because their design typically trades the expressive power for the ease of use. On the other hand, most existing database interfaces based on general-purpose query languages are difficult to use. The problem lies not only in the difficulty of learning and remembering the query language but also in the complexity of interpreting the structure and content of numerous databases distributed over the network. This dissertation considers the architecture and design of a cooperative ad hoc query interface that relieves the user of learning and recalling the query language and the underlying database.

1.1 Impedance Mismatch Between User and System

The impedance mismatch between database languages such as SQL and host programming languages has motivated much research. Deductive and object-oriented database systems have emerged to provide a uniform language for application programmers [45, 79]. While this research is expected to facilitate the development of database applications, yet another type of impedance mismatch exists between the end user's language and database languages: formal languages such as SQL burden users to learn the syntax and semantics of the language and the underlying database, and to recall them precisely at the time of query formulation. Most casual database users cannot afford time and effort for such learning. For those who can, their queries are subject to various types of failure due to the imprecise, incomplete, and incorrect nature of knowledge in human long-term memory [11, 49]. These failures include spelling mistakes, violation of language syntax and semantics, and misconception of entities and relationships in a database [55, 56, 57, 73].

The impedance mismatch faced by end users cannot be treated by assimilating the database language to the user's language alone. The so-called natural language interfaces (NLIs) are intended to provide the user's habitual language for human-computer interaction. A number of research prototypes have been developed with database systems as target applications, such as LADDER [31], PLANES [72], TQA [22], TEAM [29], and IRUS [5]. A few commercial NLI systems also exist such as INTELLECT [1], NLI DataTalker [47], and PARLANCE [4]. However, because of the difficulty of developing a large body of machine-interpretable knowledge on human linguistic behavior, these systems inevitably implement only seminatural languages and limited concepts. As a result, NLI users experience what is called proactive interference, the difficulty of remembering artificial constraints in a seminatural language [60]. Most NLIs are also limited in resolving the ambiguity and failure of unconstrained user queries. This leads to the NLI user's difficulty in debugging ambiguous and failed queries.

A field evaluation of an NLI system with some 800 BNF rules confirmed NLI user problems. In the experiment conducted by *Jarke et al* [33], NLI users performed poorly relative to SQL users. The following table summarizes the task-level performance of NLI and SQL users:

Task-Level Performance	NLI	SQL
Essentially Correct	17.1%	44.2%
Partially Solved	34.2%	23.3%
Not Solved	48.7%	32.5%

One of the concluding remarks of *Jarke et al* was that with at best 70 percent of success, both NLI and SQL interfaces are inadequate for casual database users involved in decision making.

1.2 Kaleidoscope for Controlled Near-Natural Language Interaction

Kaleidoscope is a cooperative interface which relieves casual users of the impedance mismatch that they experience in interacting with database systems [12, 13, 14]. It uses a grammar-driven menu system as a device for bridging the mismatch between the user's language and the database language. This system generates legitimate query constituents incrementally as menu choices, and users formulate a query in a sequence of point-and-click menu selections. A grammar specifying the syntax and semantics of a database language governs the system's automatic choice generation.

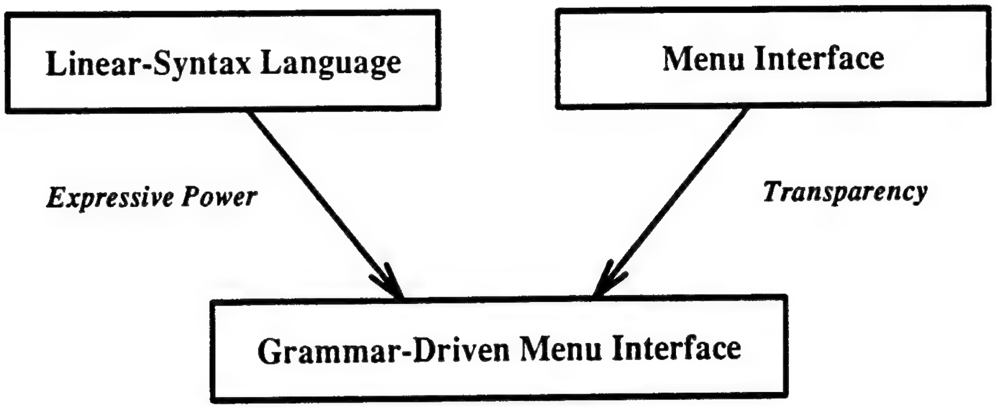


Figure 1.1: Grammar-Driven Menu Interface

As the result of combining the grammar and the menu interface, Kaleidoscope inherits the advantages of both interface types. Figure 1.1 shows the inheritance of these advantages: the expressive power from the linear-syntax language and the transparency from the menu interface. While Kaleidoscope's grammar-driven menu guidance can improve the usability of formal query languages such as SQL [12, 13], a carefully designed English-like query language (EnQL) improves the efficiency of user-system communication significantly. The following illustrates an EnQL query constructed in tens steps of menu interaction:

$$[\text{Q1}]\ \underbrace{\text{Who}}_{1}\ \underbrace{\text{wrote}}_{2}\ \underbrace{(}_{8}\ \underbrace{\text{which}}_{3}\ \underbrace{\text{'DATABASE'}}_{4}\ \underbrace{\text{books}}_{5}\ \underbrace{\text{published by}}_{6}\ \underbrace{\text{'McGraw-Hill'}}_{7}\ \underbrace{)}_{8}$$

$$\underbrace{\text{since}}_{9}\ \underbrace{\text{1982}}_{10}$$

Here, the numbers indicate the sequence of user-system interaction. EnQL supports *wh*-queries. Thus the initial menu state contains a small number of *wh*-words that have corresponding concepts in the underlying database. Subsequent menu states present only legitimate choices for extending a partial query. Kaleidoscope avoids structural ambiguity of EnQL queries by forcing the user to enclose complex phrases with a pair of parentheses (the eighth step of the query Q1).

EnQL closely follows the syntax and semantics of restricted English *wh*-queries. Thus the combination of EnQL with Kaleidoscope enables users to construct a query by recognizing a sequence of choices matching their mental query. This interface approach incurs minimal cognitive burden in the user's query formulation. Compared with the SQL version of Kaleidoscope, it relieves the user of transforming mental queries to those based on the underlying database model such as the relational model [18].

Kaleidoscope provides additional intraquery conceptual guidance to the user by building the query meaning incrementally. First, the interface guides the user's value creation by executing the partial query. This reduces the chance of extensional query failure as well as the number of choices to present on the menu for value selection. In addition, the system uses its knowledge of integrity constraints and functional dependency to avoid the semantic inconsistency of the partial query.

1.3 Technical Problems and Contributions

Guiding the user's incremental query formulation is a knowledge-based process. The *interface designer* formalizes the types of domain-specific knowledge needed for user guidance. The *interface creator* acquires such knowledge over specific databases. This acquired knowledge is interpreted by Kaleidoscope's interpreter for guiding the user to create meaningful and unambiguous queries. The system's lack of knowledge in this process results in the failure to prune irrelevant choices, which not only misleads users toward nonsensical queries but also wastes the screen space and potentially increases the user's choice search time. Normally, the benefit passes down along the hierarchy of the humans involved in the life cycle of interfaces. A good interface design benefits many interface creators; a good interface creation benefits numerous end users.

A query language conveys the underlying conceptualization of data to the user. The central theme of this dissertation is that the presence of a data model is critical in the Kaleidoscope's style of interfaces, where a menu system generates choices for reference to schema concepts. The model guides the process of interface design and creation, eventually benefiting end users. The grammar, lexicon, and query translator are model-dependent components of Kaleidoscope. The absence of an explicit model leads to the ad hoc design of these components, harming the system's transportability.

In the model-based approach, grammar design focuses on unambiguously realizing references to model concepts, taking into account the capability of the underlying query processing system. As a result, all user queries are meaningful with respect to the underlying data model. Given a grammar specification, it is possible to define a set of procedures that generate the lexicon automatically from the schema. The presence of a model also guides the implementation of a domain-independent mapping to the underlying database storage model in the query translator.

The major technical contribution of this thesis is a data model formalizing the conceptual structure of restricted English queries. Existing data models are inadequate for near-natural language interaction with database systems because of a significant conceptual gap between common English concepts and database representation of such concepts. EnQL, based on our model, enables the user to express significantly more concise queries than SQL, often by an order of magnitude. To focus on the model aspect, this thesis restricts the formal power of EnQL to the support of conjunctive queries.

The transparency of menu guidance enables the interface designer to apply the normative design principle [62] to the design of grammar-driven menu interfaces. The design process involves first setting the desired formal power of an interface as its goal. Then it explores a set of alternative designs satisfying the goal. The normative design principle requires a model for evaluating alternative designs. This dissertation presents a simple cost model of user query production as a quantitative basis of design evaluation.

The model-based, normative design approach distinguishes Kaleidoscope from conventional NLI systems, which seeks to incorporate an extensive range of syntax and semantics based on the non-normative system assumption [6, 41].

1.4 Relation to Other Work

1.4.1 Cooperative Response Approach to NLI

Past research in artificial intelligence proposed knowledge-based postquery cooperation to increase the usability of NLI systems. At the parsing level, one direction of research sought the system's robustness to extragrammatical sentences [8]. At the conceptual level, following Grice's principle of cooperation [28], so-called cooperative response systems dealt with the user's misconception about underlying information systems [21, 26, 32, 35, 34, 42, 43]. When queries fail to produce meaningful results because of the user's misconception, the system resolves specific causes of failure for the user. Yet, in this postquery cooperation approach, the system still does not use its knowledge until the user query fails.

Kaleidoscope takes a more active attitude in utilizing the system's knowledge: a system knowledgeable enough to correct or to suggest the postquery correction should use its knowledge first to guide users away from query failure. The increasing speed of computers makes it feasible for the system to take this initiative. Nevertheless, postquery cooperative response would be still needed to handle queries that have no matching tuples in the database or produce too many or too small tuples. Even in such a case, our position is to use the system's actively. Consider an extensionally failing query with $\mathcal{F}$, a set of conjuncts causing the failure. Instead of just informing the user of $\mathcal{F}$, the system suggests alternatives in query generalization focusing on this set of literals. For instance, if the keyword specification of books belongs to $\mathcal{F}$, the system suggests its generalization based on the hierarchy of keyword values. Previous research explored a range of options for such query generalization [46, 15].

1.4.2 Menu-Based NLI Approach

Recently, windows and pointing devices such as the mouse become widely available for human-computer interaction. Many screen-oriented direct manipulation interfaces have been developed, such as PICASSO [37] and PAST-3 [40]. In this type of interfaces, mouse selection and form filling are the primary means of expressing the user's intention.

As windows and pointing devices such as the mouse become widely available for human-computer interaction, Tennant and Thompson recognized that the window-based interaction could restrict the users of the so-called NLIs within the system's limited linguistic and conceptual coverage [69, 67]. This idea has developed into so-called menu-based NLI systems NLMenu [69, 71, 70], INGLISH [53, 52], and NLParse/NLGen [30]. A context-free, semantic grammar specifies dynamic choice generation in NLMenu and INGLISH. NLParse/NLGen employs a unification-based grammar to pursue linguistic generalization.

Kaleidoscope takes the notion of grammar-driven menu guidance from these menu-based NLI systems, and provides a model-based framework for the interface design and generation. A semantically rich model provides the basis for user guidance and interface design. In contrast, past research assumed a very low-level model or no explicit model at all. For example, the implicit model underlying the NLMenu grammar for relational database access [68] is not much different from the relational model [18]. As a result, NLMenu queries are often reminiscent of formal queries. The emphasis on a model in Kaleidoscope also makes it possible to provide meaning-based guidance, which previous menu-based NLI systems overlooked.

1.5 Organization of Thesis

The rest of this dissertation is organized as follows. Chapter 2 describes the features of the Kaleidoscope interface for EnQL query formulation. Chapter 3 justifies the choice of an English-like query language over the standard database query language SQL and articulates the desired features of such a query language. Chapter 4 reviews three previous grammar-driven menu interface systems. Chapter 5 presents Kaleidoscope's model-based approach and architecture. Chapter 6 describes a data model for supporting English-like queries. Chapter 7 presents Kaleidoscope's grammar formalism for choice generation. Chapter 8 presents the quantitative dimension of grammar-driven men interface design. A cost model of user query production when using grammar-driven menu interfaces is presented. Finally, Chapter 9 summarizes this dissertation.

Chapter 2

Interface for EnQL Query Formulation

ka.lei.do.scope [Gk kalos beautiful + eidos form + E -scope] an instrument containing loose bits of colored glass between two flat plates and two plane mirrors so placed that changes of position of the bits of glass are reflected in an endless variety of patterns.

– *from the on-line Webster of the NeXT computer*

Kaleidoscope bridges the impedance mismatch between an artificial query language and the user's language via a context-sensitive menu system. A grammar specifies the *pattern* of dynamic changes in the menu state. This approach enables users to formulate a query via a sequence of choice recognitions. The *bits* forming a specific menu state are derived from the underlying database. This chapter presents the Kaleidoscope interface for EnQL query formulation.

2.1 Environment and Screen Organization

Kaleidoscope takes advantage of the recent proliferation of bitmap displays and the mouse in a modern computing environment. Point-and-click selection of menu choices is the primary means of delivering the user's intention to the machine. The keyboard is occasionally used to complement the mouse input. The current implementation of Kaleidoscope runs on the XEROX LISP machine and accesses the SYBASE DBMS [64] on a remote server.

Menu choices are organized in multiple groups by their common characteristics. Each choice group is presented on a separate window. The content of a choice window depends on the state of a partial query. The system removes empty choice windows to minimize the user's visual attention space. Active windows are arranged by a prespecified order to facilitate the user's discrimination of projected target choice groups.

Two additional windows stay on the screen to guide the user's query formulation. The query status window presents the state of partial query construction. The system message window displays user-requested and system-derived intraquery information.

2.2 EnQL Features

EnQL queries begin with *wh*-words such as *who, which, where, when,* and *how.* This restriction of EnQL to *wh*-queries alleviates the potential choice explosion at the beginning of a query, where no information is available to the system other than syntactic constraints. *Wh*-words naturally factor the set of concepts qualified to appear at the beginning of sentences.

The underlying model of EnQL, as will be presented in Chapter 5 in detail, supports entities, relationships, and relationship modifiers. EnQL provides references to these concepts in the form of noun phrases, verb phrases, and adverb phrases, respectively. A common noun refers to an entity set, while a proper noun refers to an individual entity. A query may contain an arbitrary number of *wh*-words to specify the entities to be included in the output. EnQL does not include the construct for specifying projection attributes such as the SQL SELECT clause. This functionality is supported by Kaleidoscope's menu interface. We discuss this feature further in the next section.

EnQL supports both active and passive voices. A transitive verb such as "wrote" refers to a relationship between two entities, while an intransitive one such as "arrived" refers to a relationship restricting an associated entity set. An adverb phrase further restricts entity sets through the verb that it modifies. The verb "be" is used to establish the identity of entities in different entity sets.

Entities as well as entity attributes serve as the basis of prenoun and postnoun modifiers. The noun "thesis" modifying "technical report" exemplifies an entity-based modifier. EnQL also supports possessive specifiers. Thus, it is possible to express phrases such as "Gio Wiederhold's books" and "which author's books." EnQL optionally provides limited support of pronouns. A pronoun appears on the menu only when there exist entity sets realized already in the partial query that it can represent. The reference of a pronoun is resolved immediately by prompting the user with a pop-up menu of qualified noun phrases. For example, when the user selects "their," the system searches plural nouns in a partial query and presents them to the user.

English quantifiers such as "any" and "each" are not supported because they are not needed to express conjunctive queries.

2.3 Menu-Guided Query Creation

In Kaleidoscope, the user constructs a query incrementally from left to right. Figure 2.1 shows a few Kaleidoscope screen states encountered while creating the query:

$$[\text{Q1}]\ \underbrace{\text{Who}}_{1}\ \underbrace{\text{wrote}}_{2}\ \underbrace{(}_{8}\ \underbrace{\text{which}}_{3}\ \underbrace{\text{'DATABASE'}}_{4}\ \underbrace{\text{books}}_{5}\ \underbrace{\text{published by}}_{6}\ \underbrace{\text{'McGraw-Hill'}}_{7}\ \underbrace{)}_{8}\ \underbrace{\text{since}}_{9}\ \underbrace{\text{1982}}_{10}$$

A complete sequence of screen states is included in Appendix A. Each state presents only choices that are both syntactically and semantically valid for extending a partial query.

Two types of choices exist on the menu: *terminal* and *demon* choices. Most terminal choices, if selected, are appended to the partial query as they appear on the menu. Some terminal choices, however, contain ***guiding substrings***. These strings, enclosed by a pair of parentheses as in the choice "(authored) books" shown in Figure

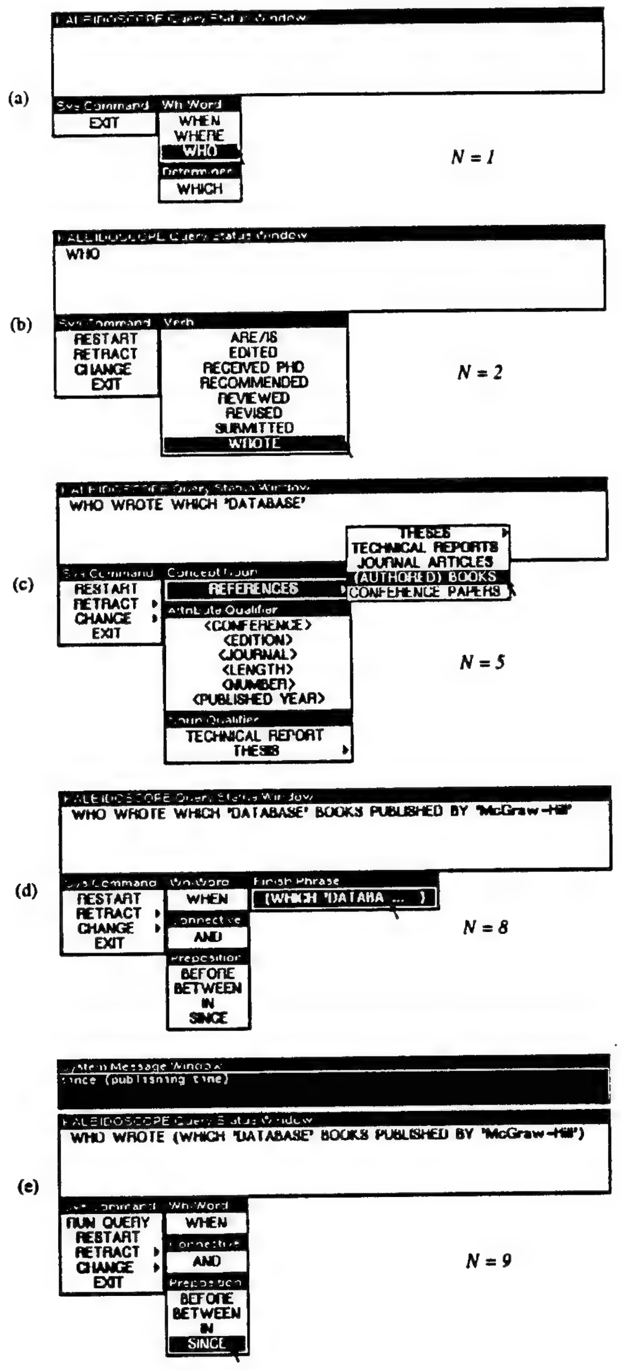


Figure 2.1: Progression of Kaleidoscope Screen States

2.1 (c), make the semantics of choices explicit to the user. The guiding substring of a selected choice is not added to the query status window. The triangle (▷) on the side of a choice indicates that the choice can be extended into submenus as in Figure 2.1 (c). Submenus organize related general and specialized terms hierarchically under a single choice.

Some menu choices provide a limited cue to the user in projecting the consequences of selecting them. English function words are representative of such choices. For example, the prepositions "at" and "on" are ambiguous to the user. Kaleidoscope associates a documentation string with each choice to help the user in projecting subsequent choice sets (Figure 2.1 (e)). An alternative approach to the choice ambiguity is to attach guiding substrings to the choice. This approach presents "since (publishing time)" instead of the choice "since" in Figure 2.1(e).

Demon choices, when selected, trigger attached actions. They are useful for guiding the user's selection of database values, such as "DATABASE," "McGraw-Hill," and "1982" in the query Q1. Those bounded by "<" and ">" in Figure 2.1 (c) are such demon choices. When selected, they prompt the user with a pop-up menu of database values or a type-in window constraining the user's input. Figure 2.2 shows a hierarchical pop-up menu for selecting the keyword value "DATABASE." Demon choices are also useful for enclosing complex phrases with parentheses to avoid potential ambiguity. We return to this point later in this section.

In addition to expanding a partial query, the user may retract and change early selections by choosing the corresponding system command choice. The user's selection of "RUN QUERY" signals the completion of a query to the system. This choice appears on the menu only when the constructed query is legitimately complete by the grammar. Often the choice set that the system produces includes a single non-command choice. The system takes such a choice automatically and proceeds to the next menu state unless "RUN QUERY" is another legitimate choice.

Value Presentation The properties of a domain determine the pop-up window type in guiding the user's value creation. For enumeratable domains such as keywords, the system prompts the user with a pop-up menu of values. In contrast, the values in

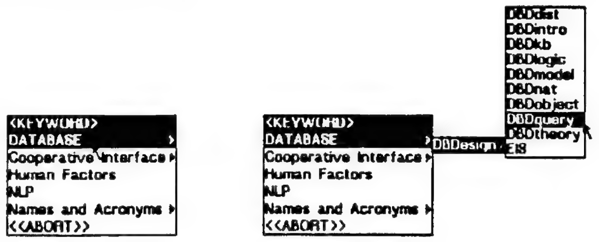


Figure 2.2: Two States of A Hierarchical Pop-Up Menu

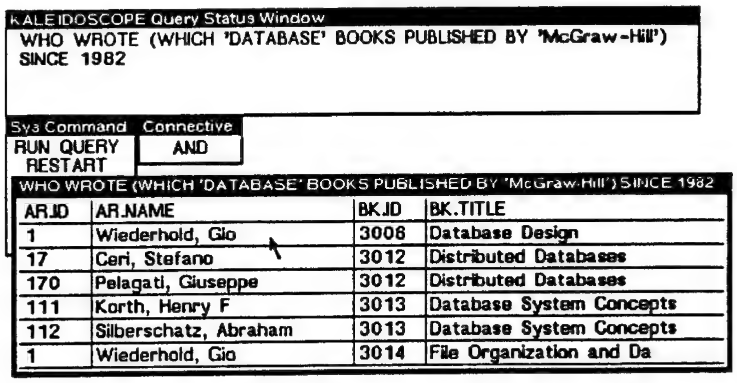
the weight domain is better not enumerated. For such a domain, the system presents a range of admissible values to the user.

Inherently hierarchical domains such as keywords are presented hierarchically to reduce the number of choices to display in the first pass. The system constructs a hierarchy for nonhierarchical domains using other related domains if their size prohibits linear listing of values. For example, the type and location of an organization is useful for forming a hierarchy of organizations.

Control of Ambiguity The ambiguity of queries expressed in an English-like language is well-known [3, 77]. Domain-specific semantics and contextual information are helpful in reducing the number of possible interpretations but do not guarantee the unique interpretation that both the user and the machine agree to.

Kaleidoscope takes the initiative in guiding users to avoid creating ambiguous queries. The menu window from which each token is selected provides the category information of the token. Overloading a choice with multiple interpretations is permitted only if the lexical ambiguity can be resolved by grammar. To avoid structural ambiguity, the system prompts users to enclose complex phrases with a pair of parentheses. For example, the query Q1, without the parentheses enclosing the object noun phrase, would be ambiguous because there are two possible interpretations on the scope of the phrase "since 1982." By providing the choice of explicitly finishing the noun phrase construction with parentheses, as shown in Figure 2.1 (d), the system avoids this structural ambiguity.

Output Presentation Kaleidoscope presents each query result in a separate spreadsheet window to facilitate further screen-based manipulation. Figure 2.3 shows such



WHO WROTE (WHICH 'DATABASE' BOOKS PUBLISHED BY 'McGraw-Hill') SINCE 1982

WHO WROTE (WHICH 'DATABASE' BOOKS PUBLISHED BY 'McGraw-Hill') SINCE 1982

AR.ID	AR.NAME	BK.ID	BK.TITLE
1	Wiederhold, Gio	3006	Database Design
17	Ceri, Stefano	3012	Distributed Databases
170	Pelagati, Giuseppe	3012	Distributed Databases
111	Korth, Henry F	3013	Database System Concepts
112	Silberschatz, Abraham	3013	Database System Concepts
1	Wiederhold, Gio	3014	File Organization and Da

Figure 2.3: Kaleidoscope Screen with Query Output

a window containing the result of the query Q1. Relational, graph-drawing, and arithmetic operations are provided as generic spreadsheet functions. With this presentation strategy, EnQL does not need features for formatting and transforming query results.

The schema of a query result consists of the attributes drawn from the entity sets specified by *wh*-words. To determine these projection attributes, the system either prompts the user with a pop-up menu of selected entity attributes or takes default projection attributes defined by the schema.

Meaning-Based Guidance In parallel with choice generation, Kaleidoscope builds the meaning of a query incrementally for further intraquery conceptual guidance.

First, by executing the partial query, the system guides the user's value creation with a dynamically computed range of values feasible for extending the partial query. This dynamic instantiation of pop-up menus not only narrows the range of choices for users but also reduces the chance of extensional query failure – the failure of syntactically well-formed queries to produce tuples due to the user's misunderstanding of database contents [35].

While the extensional failure of a query is reduced by guiding the user to select only values feasible to extend the partial query, the so-called intensional failure [42] is still possible to occur. In the next section, we present an intraquery conceptual guidance scheme for avoiding intensional query failure.

2.4 Intraquery Conceptual Guidance

A query may contain semantic redundancy in its condition without the user's notice. A syntactically well-formed query may fail semantically if its hidden semantic redundancy leads to an inconsistent query condition. For users, this type of failure is more difficult to deal with than syntactic failure because the cause of failure may not be obvious.

This section describes the heuristics used in Kaleidoscope to guide users away from intensional failure. The system's stored integrity constraint rules and functional dependency play the key role in this intraquery conceptual guidance.

2.4.1 Rule-Based Guidance

Integrity constraint rules express the semantic invariant of database extensions. They are useful for checking the validity of update transactions and transforming queries to semantically equivalent yet less expensive ones in evaluation [38] and in constraint validation [54]. They are also useful for guiding users away from misconception. IN Kaleidoscope, by checking the meaning of a partial query against the system's knowledge of integrity constraint rules, the system detects user misconceptions in the middle of query composition. Consider a query

[Q2] $\underbrace{\text{Which}}_{1}\ \underbrace{\text{instructors}}_{2}\ \underbrace{\text{who}}_{3}\ \underbrace{\text{taught}}_{4}\ \underbrace{\text{CS445}}_{5}\ \underbrace{\text{are}}_{6}\ \underbrace{\text{students}}_{7}\ \underbrace{\text{who}}_{8}\ldots$

to a university database with an integrity constraint:

[IC1] Student instructors never teach CS 400 or higher level courses.

After the user's seventh selection, the system recognizes that the query Q2 becomes inconsistent with the integrity constraint, and warns the user of the inconsistency. This early detection of misconception, compared with postquery detection [26], saves the user's effort that would otherwise be wasted on completing a query bound to produce no meaningful result.

The same integrity constraint is also useful for generating informative messages. Consider IC1 rephrased as follows:

[IC2] If an instructor teaches a CS course whose number is higher than or equal to 400, then the instructor is not a student.

Once the user finishes the fifth selection in the query Q2, the system derives a constraint that the instructor is not a student. This derived constraint is useful for guiding the user away from potential semantic inconsistency in the user's subsequent selections. However, in general, presenting all derived information may distract experienced users needlessly. The user has an option of disabling the system's presentation of derived informative messages.

Information Presentation Threshold The definition of an integrity constraint rule assigns one of three level indicators to each derived literal:

Level 1: obvious.

Level 2: informative.

Level 3: critical.

We may think of the query condition selected by the user corresponding to the level 0. A more fine-granuled classification of derived information is conceivable. For example, the level 2 may be divided into two levels of more informative and less informative. The user may choose one of these levels as the threshold of intraquery conceptual guidance. For example, if the user chooses the level 2, the system presents the derived information of levels 2 and 3 to the user. The default threshold level is set to 3, so that only critical derived information such as integrity constraint violation is presented to the user. Literals of the level 1, despite their obviousness, still need to be derived because they may be called in a chain of inference for deriving other critical and informative literals.

2.4.2 Guidance on Attribute Modification

Functional dependency is important in avoiding redundant attribute modification of entities. Kaleidoscope applies the following heuristics:

- If an attribute is selected for restricting an entity set, it is pruned later on. The system assumes that the user does not modify an attribute repeatedly. For example, the menu state of Figure 2.1 (c) does not show the choice "<keyword>" because it was selected in the previous step. (See the menu state for $N = 4$ in Appendix A.) The set of comparators in EnQL, which includes "between" and "not between," enables the user to express an arbitrary range in one-time attribute selection.

- Consider a functional dependency from the attribute A to the attribute B ($A \rightarrow B$). If the user's added predicate binds A to a constant, the system also prunes A's dependent field (B) in the conjunctive extension of the partial query. For example, consider a partially constructed noun phrase "Which employees whose id is 1356 ..." Since the employee id is the key attribute of the entity *Employee*, the user is not allowed to add further attribute-based restrictions to "employee."

- If the key attribute A is set to a constant after its dependent attribute B (e.g., "employee department") is specified, the system informs the user of the redundant restriction on B, and suggests the user to remove redundancy to avoid potential query failure.

- If the user's selection derives a constraint that a field C is restricted to a range of values, this range guides the user's further specification of C.

2.5 Summary

This chapter presented the features of Kaleidoscope's interface for EnQL query formulation. The system's guidance is summarized as proposing legitimate EnQL constituents as menu choices step by step and providing immediate semantic feedback to the user. This interface approach enables the user to construct a query via a sequence of choice recognitions, thus freeing the user from the burden of learning and recalling precisely the query language and the database. The system uses its knowledge of integrity constraints and functional dependency to avoid conceptual query failure. Restricting the user's value creation to the result of executing the partial query reduces the chance of extensional query failure.

Chapter 3

Why An English-like Query Language?

As Kaleidoscope combines the menu interface and the linear-syntax language, it inherits the merits of both. Its interface is transparent to the user while offering the expressive power of a linear-syntax language. Kaleidoscope's grammar-driven menu interface approach is applicable not only to an English-like query language but also to formal languages as shown by our early work on menu-guided SQL interface [12, 13].

This chapter presents an argument of why an English-like query language is beneficial for end users compared with SQL and articulates the desired features of such an English-like language.

3.1 EnQL vs SQL

EnQL enables the user to express queries more concisely than SQL, and relieves the user of transforming mental queries to those based on the underlying database implementation.

3.1.1 Previous Result

The results of previous human factors studies comparing restricted NLIs with formal language interfaces are in general negative on NLIs [33, 60]. There is, however, a positive side on the use of restricted natural language for querying database systems. Jarke *et al* [33] found in their field experiment comparing an NLI with the SQL interface:

- NLI queries consume about one third the number of tokens than SQL queries (10.64 vs 34.19 tokens).
- NLI users spent about 29 percent less time, on the average, to formulate and execute a query than SQL users (7.7 min vs 10.8 min).

Although this gain in the per-query efficiency is offset by 50 percent more queries issued by NLI users per task (8 English queries vs 5.1 SQL queries per task), the statistics measured by Jarke *et al* indicates that restricted English queries are easier and more efficient to phrase than SQL queries.

3.1.2 EnQL Case

Our experience of EnQL over the bibliographic application shows even a stronger result than the findings of Jarke *et al* in favoring the use of restricted English query language over SQL. Some examples show that SQL translations of EnQL queries may consume ten times as many tokens as EnQL queries and involve joins of several tables. For example, while the query Q1, created in ten steps, consumes nine tokens, its SQL translation comprises 89 tokens. Figure 3.1 shows a pair of EnQL and SQL queries. Figure 3.2 presents the underlying database schema based on the structural data model [75]. In counting tokens, we have excluded parentheses and punctuation marks ("." and ","). One EnQL token 'DATABASE' alone accounts for 20 SQL tokens (2 table declarations, 2 joins, and 1 restriction).

The number of tokens required to express a query is critical to the performance of query interface users. In the absence of other information on the complexity of a

[EnQL]

$\underbrace{\text{Who}}_{1}\ \underbrace{\text{wrote}}_{2}\ \underbrace{(}_{8}\ \underbrace{\text{which}}_{3}\ \underbrace{\text{'DATABASE'}}_{4}\ \underbrace{\text{books}}_{5}\ \underbrace{\text{published by}}_{6}\ \underbrace{\text{'McGraw-Hill'}}_{7}\ \underbrace{)}_{8}\ \underbrace{\text{since}}_{9}\ \underbrace{\text{1982}}_{10}$

[SQL]

```
SELECT  author_reference1.author,
        person2.pname,
        book9.id,
        reference7.title,                                           9
FROM    book book9,
        organization organization10,
        publish_reference publish_reference11,
        keyword_view keyword_view4,
        reference_keyword reference_keyword5,
        reference reference7,
        author_reference author_reference1,
        person person2                                              17
WHERE   publish_reference11.reference = reference7.id and
        organization10.id = publish_reference11.organization and
        book9.id = publish_reference11.reference and
        book9.id = reference7.id and
        book9.id = reference_keyword5.reference and
        keyword_view4.id = reference_keyword5.keyword and
        author_reference1.reference = book9.id and
        author_reference1.author = person2.id and
        organization10.name = 'McGraw-Hill' and
        reference7.year >= 1982 and
        keyword_view4.string = 'DATABASE'                           63
                                              Total token count     89
```

Figure 3.1: SQL Translation of Q1

Kaleidoscope's query translator makes no attempt of minimizing the token count. Thus its SQL translation may include extraneous tokens and conditions. However, queries produced by the human are equally likely to have such extraneousness. In this example, range variable declarations are extraneous because tables are all distinct. In addition, the WHERE clause has hidden redundancy: Either the first or the third condition may be removed.

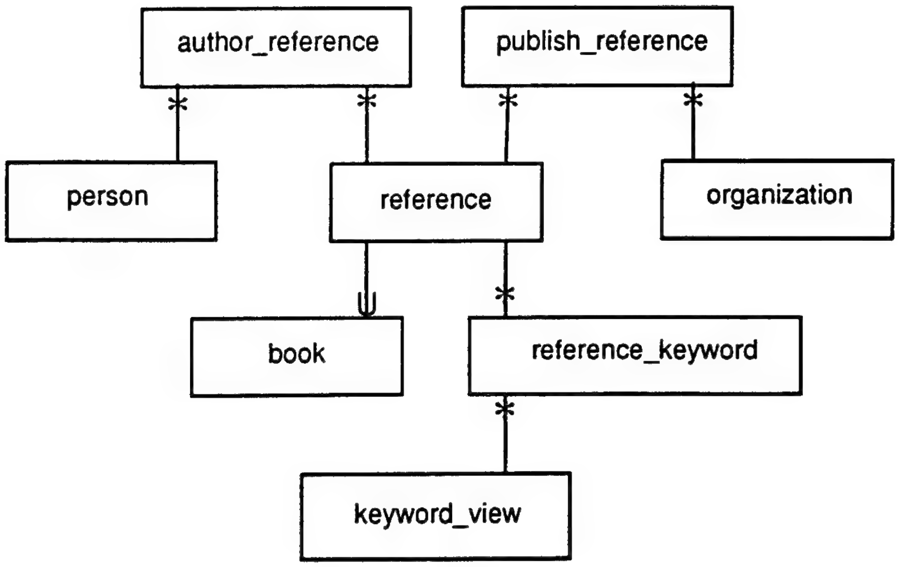
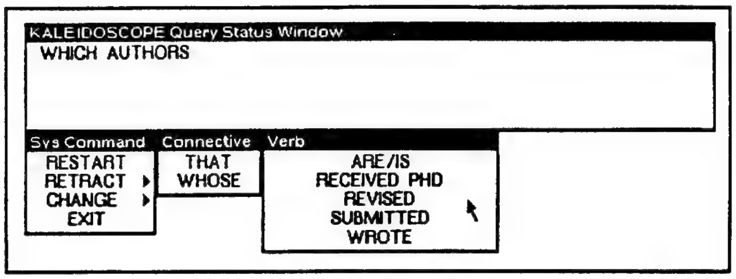


Figure 3.2: The Database Schema Covered By The Query Q1

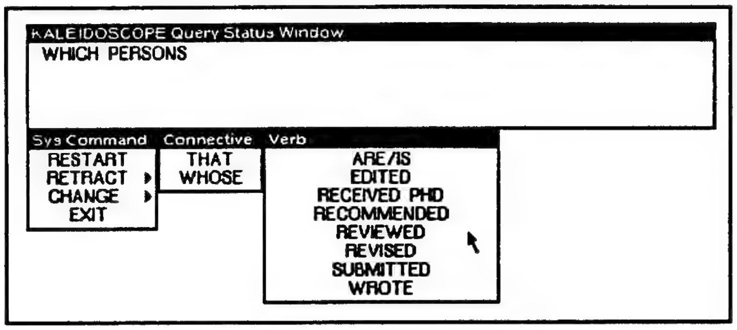
$\boxed{X}$ —*— $\boxed{Y}$ represents one-to-many ownership connection from X to Y.
$\boxed{X}$ —⊃ $\boxed{Y}$ represents a subset connection from X to Y.

query, the ratio of the required number of tokens and the capacity of human short-term memory (7 ± 2 [44]) could be a measure of the user's cognitive burden. The higher this ratio, the more cognitive swapping, we suspect, is required to produce a query. Note that the EnQL query Q1 roughly fits into human short-term memory, while the its SQL equivalent does not.

Automatic Database-Level Join Generation The SQL query in Figure 3.1 involves the join of eight tables – an indication of the degree of conceptual complexity that the user might have to go through if the query had to be written manually. Human factors experiments have been conducted on SQL and other formal languages in their early development stage [56]. In the experiment conducted by Welty and Stemple [73], SQL join was found to be one of the two problematic features of SQL. (GROUP BY is the other, and the most problematic SQL feature.) Considering that the join queries in this experiment typically involved two tables, we can easily project the difficulty of phrasing SQL queries involving several joins.



(a) After selection of a specialized term



(b) After selection of a general term

Figure 3.3: Illustration of Commitment Degree Choices

EnQL users do not have to worry about the conceptual burden of joining. The system automatically creates an SQL query involving the join of multiple database tables. As the detail of this translation will be given in Chapter 5, each translated SQL query corresponds to a minimal dynamic view in the sense that only tables necessary to answer a given query are joined.

3.2 Desired Features of EnQL

The fact that EnQL queries consume a significantly smaller number of tokens than its SQL translations suggests that EnQL queries are more efficient and probably easier to phrase than SQL queries. This leads us to ask what the elements of the English-likeness are that contribute to the conciseness of EnQL queries, and that will be of further benefit to grammar-driven menu interface users. Our answer to this question is summarized in terms of four degrees of freedom:

1. *Distribution of modifiers* over the span of a sentence: The underlying data model should support a rich set of modifier types. This set includes verb modifiers as well as prenoun and postnoun modifiers.

2. *Reduction of query production steps*: For example, the user should be able to choose a shorthand expression "DATABASE books" instead of its full-fledged version "books written on DATABASE." The latter structure is still needed for the user to add adverb phrases modifying the verb. Similarly, the phrase "McGraw-Hill's books" is a shorthand for "books published by McGraw-Hill."

3. *Alternative ordering of references*: While users can tolerate limited syntax with menu guidance, too restrictive a syntax forces users to navigate through a narrow network of choice sets. To lessen this burden, the language syntax should support means of alternatively ordering references to entities and relationships. The particular syntactic features useful for this purpose are passive voice and adverb preposing. These enable the user to choose the logical object and an adverb phrase of the verb phrase at the beginning of a query, respectively.

4. *Choice of commitment degree*: The user's selection of a generalized term, compared with a specialized term, makes a weak commitment in referring to entity sets and relationships, thus leaving more options in the subsequent menus. For example, the choice set following "which persons" includes verbs that are not applicable to "which authors," such as editing books. Figure 3.3 shows two Kaleidoscope screen states for comparison.

Chapter 4

Previous Work

As windows and pointing devices such as the mouse become widely available for human-computer interaction, Tennant and Thompson recognized that the window-based interaction could restrict users within the system's limited linguistic and conceptual coverage [69, 67]. This idea has developed into the so-called menu-based NLI systems NLMenu [69, 71, 70], INGLISH [53, 52], and NLParse/NLGen [30].

Taking the notion of grammar-driven menu guidance from these menu-based NLI systems, Kaleidoscope provides a model-based framework for the interface design and generation. Before introducing Kaleidoscope's approach, this chapter reviews menu-based NLI systems and discusses their limitations.

4.1 Architectures

4.1.1 NLMenu/INGLISH

NLMenu and INGLISH capture constraints on choice generation in the so-called semantic grammar. Proposed by Burton [7], this grammar formalism represents both syntactic and semantic constraints uniformly in a collection of rewrite rules. Interesting concepts in the application domain are chosen as grammar symbols. NLMenu and INGLISH use a context-free version of semantic grammar. Figure 4.1 shows such a grammar for the bibliographic database application. The rule SG1 states that a

SG1	S	→	AUTHOR-NP, AUTHOR-VERB, AUTHORED-OBJ-NP, {? AUTHOR-ADVPS}
SG2	AUTHOR-ADVPS	→	AUTHOR-ADVP, {? AUTHOR-ADVPS}
SG3	AUTHORED-OBJ-NP	→	BOOK-NP
SG4	AUTHORED-OBJ-NP	→	JOURNAL-ARTICLE-NP
SG5	AUTHORED-OBJ-NP	→	THESIS-NP
SG6	THESIS-NP	→	PHDTHESIS-NP
SG7	AUTHOR-ADVP	→	AUTHORING-TIME-ADVP
SG7	AUTHOR-ADVP	→	AUTHORING-KEYWORD-ADVP

Figure 4.1: A Context-Free Semantic Grammar

query is composed of a noun phrase representing the author (AUTHOR-NP), a verb (AUTHOR-VERB), another noun phrase representing the authored object (AUTHORED-OBJ-NP), and an optional list of adverb phrases (AUTHOR-ADVPS). Those inside {? —} form an optional sequence. The rule SG2 governs recursive instantiation of adverb phrases (AUTHOR-ADVPS) based on a single adverb phrase (AUTHOR-ADVP). Rules SG3, SG4, SG5, and SG6 capture the generalization/specialization (*ISA*) hierarchy of authored objects.

Context-free semantic grammar enables the early activation of semantic constraints using a simple predictive context-free grammar interpreter. However, this approach is bound to show limited context-sensitivity in user guidance:

1. *Unconstrained Recursion:* Recursion in grammar represents the repeated occurrence of patterns in sentences. Context-free recursion such as the rule SG2 of Figure 4.1 lacks context-dependent constraining power. Thus the system repeatedly generates choices whose semantic basis has been specified by the user's previous selections. For example, the rule SG2 enables the user to construct multiple time adverb phrases in a single query as in "Who wrote books in 1983 in 1985?" If the query translator interprets two adverb phrases as a conjunction, the publishing time of books is restricted to two disjoint ranges in the translated query. No output tuples result from this redundant query. The user, however, may have a different, disjunctive interpretation from the user.

2. *Coarse Semantic Granularity:* The difficulty of developing and maintaining semantic grammar over large application domains [65, 66] leads to a grammar often written with coarse semantic granularity. For example, the rule SG1 relates AUTHORED-OBJ-NP with AUTHOR-ADVPS. This rule assumes that a single set of adverb phrase types apply to all authored object types. While this could have been true in the designer's initial conception, it is no longer valid as the system's semantic coverage expands to include additional authored object types, such as journal article. Authoring journal articles may accompany adverb phrases such as "in IEEE transactions" for specifying the journals in which articles appear. This type of adverb phrases is not applicable to other authored object types.

Semantic grammar is not reusable. To overcome this problem, NLMenu provides a template grammar so that the interface creator compiles a run-time semantic grammar by combining this template grammar with the so-called *portable spec,* a collection of domain-specific information.

4.1.2 NLParse/NLGen

NLParse/NLGen is an interface to Prolog knowledge base. To avoid semantic grammar problems, NLParse/NLGen employs a unification-based grammar which augments linguistic categories with Prolog terms representing syntactic and semantic features. Lexicon supplies the run-time binding of these features. Unification of semantic feature bindings enforces domain-specific constraints. For example, the verb "schedule" is restricted to take only event nouns such as "talks" for its object, and talks may be scheduled for "interviewers." NLParse/NLGen deductively generates lexicon entries dynamically from the assertions in its knowledge base. While this is a noble idea, it does not show how complete and general such deduction could be given an arbitrary knowledge base structure. NLParse/NLGen only takes advantage of context-sensitivity to a limited degree. No attention has been paid to pragmatic constraints such as avoiding unconstrained recursion.

4.2 Absence of High-Level Data Model

4.2.1 Influence of NLI Approach

The NLI approach [6, 9, 41] is pervaded by the goal of building non-normative systems capable of recognizing unconstrained natural language input. This goal has led NLI research to put great emphasis on syntax to build a general-purpose linguistic processor. No assumption is made on the underlying data model in grammar development. To represent the meaning of natural language input, NLI systems commonly employ an intermediate-level knowledge representation. At the application development time, a loose connection is established from the general linguistic module to the capability of the underlying query system. The emphasis on the surface-level capability and the adoption of the top-down layered mapping inevitably introduce interlayer capability mismatch. As a result, the meaning of some parsed sentences may not be represented internally, and those whose meanings are known to the system may not be processed by the underlying query processing system.

The so-called menu-based NLIs, despite their use of menus to control user input, inherit the top-down perspective of NLI design and processing. Grammar is designed without much concern on the formal model of the underlying system capability, often overlooking some crucial aspects of formal language design. The absence of pronoun support in these systems exemplifies this. As the means of referring to entities in a query, pronouns play the role of tuple variables in relational query languages. While these systems may be excused for not supporting intersentential anaphora, leaving out intraquery anaphora makes their query language relationally incomplete [19]. Queries such as "Which students earn more salary than *their* advisors?" are not expressible in these systems, although they are in relational languages. The lack of concern on formal expressive power in surface language design leads to an additional problem: the power of a surface query language may exceed the capability of the underlying system. Thus, the system may guide users to create queries that it can parse but not necessarily process them to return meaningful results.

4.2.2 Problems with Low-level Implicit Data Model

The style of NLMenu queries is reminiscent of formal language queries [61]. We consider that this is largely attributed to an implicit, low-level data model underlying the NLMenu interface for relational database access [68]. The level of abstraction captured in NLMenu's portable spec is not much different from that of the relational model [18]. Tables and fields are defined as two separate categories of common nouns. English terms are specified for all possible join paths. Tables may be restricted and joined via postnoun modifiers. The following query shows an NLMenu approximation of the EnQL query Q1.

[NLMenu-Q1]

$$\underbrace{\text{Find}}_{1}\ \underbrace{\text{persons}}_{2}\ \underbrace{\text{who wrote}}_{3}\ \underbrace{\text{books}}_{4}\ \underbrace{\text{published by}}_{5}\ \underbrace{\text{'McGraw-Hill'}}_{6}$$
$$\underbrace{\text{and}}_{7}\ \underbrace{\text{whose book published year is}}_{8}\ \underbrace{\text{greater than}}_{9}\ \underbrace{\text{1980}}_{10}$$
$$\underbrace{\text{and}}_{11}\ \underbrace{\text{whose book keyword is}}_{12}\ \underbrace{\text{'DATABASE'}}_{13}$$

This query is only an approximation of the EnQL query Q1 because NLMenu grammar does not allow the result of a join query to be drawn from more than one table.

The EnQL query Q1 is more concise and comprehensible than the above NLMenu query. The difference is attributed to the rich set of modifiers distributed over the span of the query Q1. While NLMenu-Q1's modifiers are all postnoun, Q1 has a prenoun modifier, a postnoun modifier, and an adverb phrase. The freedom of distributing the modifier load functionally constitutes a key to constructing concise and comprehensible queries. Although NLMenu grammar can be extended to incorporate prenoun modifiers, its underlying model does not permit adverb phrases. To support adverb phrases, the model should include two additional conceptual primitives, one for representing verbs and another for verb modifiers. The verb phrases appearing in NLMenu queries are not supported as full-fledged concepts. They are just connectors specifying table joins. For example, the third and the fifth choices of the query Q1 specify the following join paths:

"who wrote"	: person → author_reference → book_reference.
"published by"	: book_reference → publish_reference → organization.

Here, book_reference is a view created by joining two tables "book" and "reference." With no conceptual distance between NLMenu queries and relational queries, the process of query translation is straightforward. Relational queries are directly composed from the parse trees of NLMenu queries.

4.2.3 Loose Connection Between Data Model and Language

The loose connection between the data model and the grammar leads the interface designer to overlook important model features useful for user guidance. The *ISA* hierarchy is such a model feature. It is useful for structuring information economically and for saving computation if embedded in the inference (e.g., [2]). In the grammar-driven menu interface, it is the basis of providing the user with the choice of commitment degree, which we discussed as a desired EnQL feature in the previous chapter. In NLParse/NLGen, although its knowledge base represents the *ISA* hierarchies, only terms corresponding to the leaf nodes of hierarchies are presented to the user.

INGLISH also does not convey the type hierarchy of SmallTalk objects to the user. Furthermore, it shows how a grammar created in an ad hoc fashion is troubled with an ad hoc, domain-specific query translation scheme [53].

4.3 Summary

In this chapter, we have discussed the problems associated with the approaches taken by previous grammar-driven menu systems. In the next chapter, we present the model-based approach and architecture of Kaleidoscope.

Chapter 5

Model-Based Approach

I don't think that simple home appliances — stoves, washing machines, audio and television sets — should look like Holliwood's idea of a spaceship control room.

– Donald Norman, *The Psychology of Everyday Things* (1988)

5.1 Introduction

To support Kaleidoscope's style of user-system interaction, coupling of syntactic, semantic, and contextual information is indispensable. The lack of semantic and contextual information in choice generation results in the failure to prune irrelevant choices, which not only misleads users toward nonsensical queries but also wastes the screen space and potentially increases the user's choice search time. On the other hand, in defining the interface architecture, there is a seemingly conflicting objective of facilitating specific interface creation. A high degree of modularity is required to meet this objective. Specifically, it is desirable for the architecture to provide (1) a domain-independent grammar, (2) a domain-independent translator, and (3) ease in generating a domain-specific lexicon, where the lexicon refers to a collection of categorized choices.

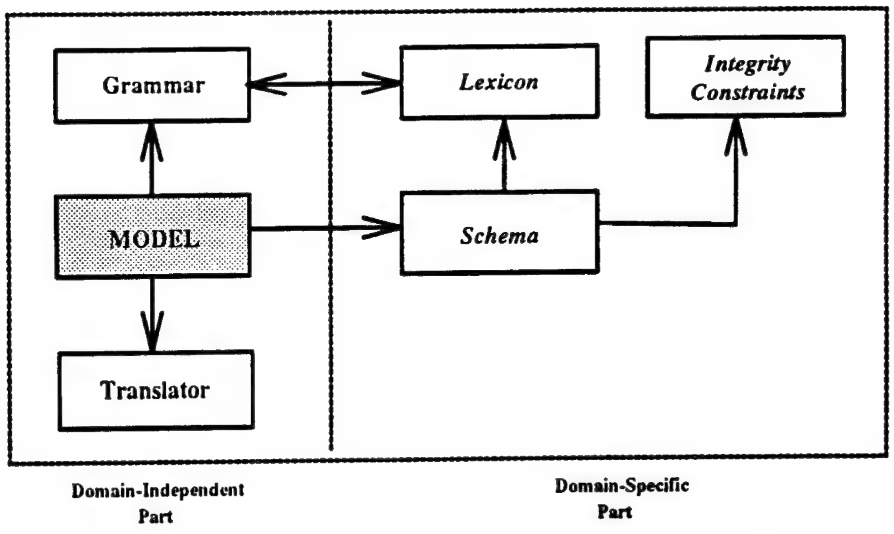


Figure 5.1: Model-Based Approach

The central theme of this dissertation is that in seeking these architectural goals, the presence of a high-level data model is critical. The absence of an explicit model leads to ad hoc grammar design and query translation, thus harming the transportability of the system. Existing data models are inadequate for supporting the desired features of EnQL. There is a significant conceptual gap between common English concepts and database representation of such concepts. Ignoring this gap would either force users to create cumbersome queries, or overload the grammar with a complex mapping to achieve a comfortable level of English-likeness.

5.2 Model-Dependency

Recognizing the conceptual distance between English-like queries and the the underlying database representation, this thesis defines a data model formalizing the conceptual structure of EnQL queries. Figure 5.1 shows how this model serves as the basis of defining the run-time components of the transportable grammar-driven menu system. First, it guides the the acquisition of domain-specific information in the schema. Integrity constraint rules are expressed in terms of this schema.

In the model-based approach, grammar design focuses on the specification of rules for realizing unambiguous and meaningful references to model concepts and constructing the query meaning incrementally. Unlike the conventional NLI approach in which the goal of non-normative systems is pervasive [6, 41], the normative design principle [62] is applied. The design process sets a target expressive power by considering the capability of the underlying query processing system. Alternative designs are evaluated by a cost function. In our research, we have taken conjunctive queries as the target expressive power, and devised a simple cost model of user query production when using grammar-driven menu interfaces. One benefit of this model-based grammar design is that all queries created via menu guidance are meaningful with respect to the data model.

Grammar design produces a set of preterminal category definitions. Instead of acquiring the domain-specific lexicon entries independently, the model-based approach generates the lexicon automatically from the schema. A set of model-dependent yet domain-independent procedures implements this automatic lexicon generation. For this automated lexicon generation, a schema definition includes a collection of English words/phrases as references to schema concepts. This approach first insures the semantic consistency between the schema and the lexicon. Second, it relieves interface creators of dealing with the linguistic part of the interface. Finally, with a well-defined model, it is possible to define a mapping from this model to a target storage model. This thesis takes the relational model for the target model. In addition, we limit the expressive power of EnQL to conjunctive queries. This restriction avoids the quantifier scope resolution problem of true English, and thus enables incremental forward-chaining inference on the partial query meaning.

5.3 Kaleidoscope Architecture

Figure 5.2 shows the architecture of Kaleidoscope. Rounded boxes represent the knowledge structures, and rectangles represent domain-independent procedures. Arrows indicate the direction of information flow.

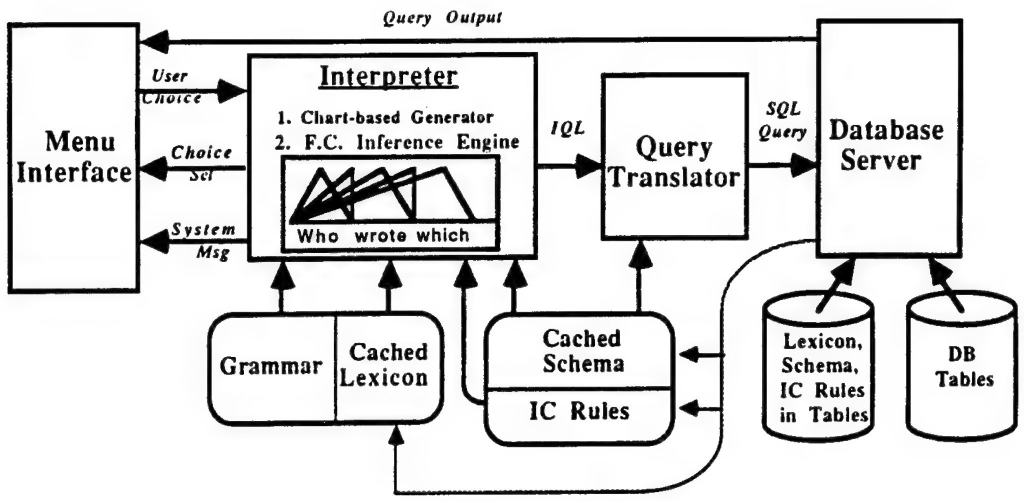


Figure 5.2: Kaleidoscope Architecture

5.3.1 Grammar and Lexicon

This section briefly describes Kaleidoscope's grammar formalism. A more detailed description will be presented in Chapter 7.

Around 40 phrase structure rules specify EnQL. The following shows a top-level rule prescribing the construction of queries comprising a noun phrase (**NP**) followed by a conjunction of verb phrases (**VPS**):

S	$\longrightarrow$	NP	subj_entity, subj_evar, (detl *:init* 'wh), relationship, _rel_obj, number, (case *:init* 'subj), compare_pred,	VPS	subj_entity, subj_evar, case, relationship, number, (form *:init* '[pres, past]), _xmodifiers

Each grammar symbol is augmented by a collection of feature attributes (shown in boxes next to symbols) that formalizes the context of constituent structures. Both syntactic and semantic features are captured this way. The run-time binding of these features comes primarily from the lexicon, although grammar rules often provide domain-independent values such as `wh` and `subj`. Feature attributes may take a limited constraint formula: disjunction of atoms (enclosed by "[" and "]"), negated disjunction of atoms, or conjunction of one disjunction and one negated disjunction. Unification of feature bindings is enforced between a parent rule and its children to block unnecessary application of child rules.

Kaleidoscope also supports attachment of several types of procedural decoration to grammar rules. These decorations enable interface designers to capture arbitrary constraints and actions in grammar rules.

The lexicon consists of a list of preterminal categories. Each category defines a list of feature attributes, a list of choices, and a display menu window. Table 5.1 shows sample lexicon entries. Semantic feature attributes, such as **entity**, **v_subj**, and **v_obj**, refer to the schema concepts.

5.3.2 Schema and Integrity Constraint Rules

The schema defines domain-specific concepts in frames. Each frame contains information on mapping to underlying databases. Schema concept names occur as the values of semantic features of lexical items. Integrity constraint rules are production rules based on the schema concepts.

5.3.3 DBMS-based Management of Knowledge Structure

Complex dependencies exist among schema entries and from schema to integrity constraint rules. For example, renaming an entity requires updating more than dozen places in the underlying knowledge structure. File-based storage is inadequate for keeping track of such complex dependencies. Kaleidoscope manages the schema, integrity constraints, and the English terms for reference to schema terms in the relational database. This DBMS-based approach first provides the locality of changes and supports set-oriented queries to the knowledge structure. Furthermore, with rules

WH-PN:	
“who”	
entity =	*Person*
v_subj =	[*Author_Reference, Edit_Book, ...*]
v_obj =	NIL
V:	
“wrote”	
rel =	*Author_Reference*
subj_entity =	*Author*
obj-entity =	$Reference \wedge \overline{Edited\_Book}$
arity =	2
tense =	past
form =	past
ENTITY-SET-N:	
“books”	
entity =	*Book*
v_subj =	NIL
v_obj =	[*Author_Book, Edit_Book, Publish_Book*]
countp =	plus
number =	pl
“(authored) books”	
entity =	*Authored_Book*
v_subj =	NIL
v_obj =	*Author_Book*
countp =	plus
number =	pl

Table 5.1: Sample Lexicon Entries

and triggers supported as a part of DBMS functionality [74, 64, 63], updates can be automatically propagated based on known dependencies.

Over thirty relations are used to represent the schema, integrity constraint rules, and lexicon. The information in these relations is then cached in the main memory for run-time efficiency. Shaded arrows in Figure 5.2 indicate that the structure at its source is cached at the destination. This caching is more than simple duplication. A set of procedures generates schema frames, rules, and lexical entries in the form desired by the interpreting procedures [76]. The use of relational DBMS also facilitates creation of interfaces covering a subset of concepts in an application domain.

5.3.4 Interpreter

For the interface creator, it is desirable for the interpreter of a complex knowledge structure to exhibit simple behavior so that the consequence of changing the structure is easily projectable. Kaleidoscope's interpreter is made of two interacting procedures:

- *Grammar interpreter* incrementally generates choices as specified by a grammar based on the partial query state. This interpreter is based on the notion of active chart, a run-time structure for keeping track of partial query creation. The unification embedded in this interpreter is extended to recognize the generalization/specialization hierarchy. Although the partial structure handled by this unification is not as sophisticated as those in other unification-based grammar formalisms, this extension is original.

- Forward-chaining inference engine keeps track of the partial query meaning and generates informative or corrective messages. Because of the forward-chaining nature, Kaleidoscope uses OPS5 [25, 20].

5.4 Overview of Processing

Given a partial query, the interpreter generates a set of lexicon match descriptors (LMDs). An LMD is specified by:

- A preterminal category symbol,
- A list of feature attribute constraints
- A filtering constraint expressed in terms of feature attributes

For each LMD, the system retrieves a list of matching items from lexicon, and sorts and presents them on the screen. Some choices on the screen may actually represent multiple lexicon entries. Once the user selects a choice from the screen, the system expands its hypothesis on the partial query, which is internally represented by an incomplete forest (a set of trees) sharing intermediate and leaf nodes, and generates a new set of LMDs.

To illustrate, consider a partial query consisting of a single user-selected choice "which." An incomplete forest maintained by the system leads to the generation of "authors" as one of choices for its extension. Once this choice is selected by the user, the syntactic and semantic information associated with "authors" are used to extend the hypothesis forest. Since the noun "authors" refers to the entity set *Author*, this information is used to select subsequent verb phrases. This cycle goes on until the user selects the finishing choice "RUN QUERY" enabled when it is legitimate to finish.

5.5 Summary

This chapter has presented Kaleidoscope's model-based approach to the design of a tightly constrained transportable grammar-driven menu system. The next chapter describes the data model for supporting this model-based approach.

Chapter 6

EnQL Data Model

Many readers, I suspect, will take the title of this book as suggesting that women, fire, and dangerous things have something in common – say, that women are fiery and dangerous. Most feminists I've mentioned it to have loved the title for that reason, though some have hated it for the same reason. But the chain of inference – from conjunction to categorization to commonality – is norm. The inference is based on what it means to be in the same category: things are categorized together on the basis of what they have in common.

– Georgy Lakoff, *Women, Fire, and Dangerous Things* (1987)

6.1 Introduction

Past research in natural language processing attempted to capture the semantic roles of noun phrases (NPs) in a small number of deep cases such as agent, patient, and instrument [24, 58]. The intent is to use these cases as the basis of specifying the argument structure of verbs and representing the meaning of sentences for general inference. For example, the definition of the verb "wrote" includes two mandatory cases, agent and patient. An NP that refers to a set or an individual of *Author* fills the agent case. Similarly, an NP that refers to the entity *Reference* fills the patient case. The verb "wrote" may have optional cases relating it to the time, place, and keyword areas of authoring. Rules are needed to assign possible NP positions, such as subject, object, and prepositional phrase (PP) object, to deep cases.

While the notion of deep case engenders generality in representing sentence meanings, it is difficult to agree on what constitute a complete collection of deep cases. In the problem domain of querying databases, fortunately, such generality is not needed because the semantics of a specific database determines the collection of meaningful verbs and their argument structures. This chapter develops a model to be used as the basis of representing this database semantics.

6.1.1 Concerns in Data Model Development

A significant conceptual gap exists between EnQL and the database representation of entities and relationships. Thus the development of a model should address the issue of what level of representation to choose and how to resolve this conceptual gap. In Kaleidoscope, the grammar and the query translator together take the burden of resolving this conceptual gap. If one of these components takes less burden, the other is subject to more. Our approach is to relieve the grammar of the burden as much as possible so that the grammar embodies a simple mapping between EnQL and the model. Thus the model captures the level of representation that is close to the conceptual structure of restricted English queries. The query translator implements a mapping between this model and the underlying storage models such as the relational model [18]. Our premise for this decision is that once a mapping to a storage model is defined, its benefit persists. On the other hand, we expect many variations of EnQL grammar. The simplicity of the mapping between EnQL and the data model facilitates the development of these variations. In addition, the simple mapping between EnQL and the model makes the process of instantiating specific database interfaces transparent. The interface creators easily project the consequence of introducing new concepts in the schema.

Our secondary concern is on the representation of the information needed for enforcing the integrity of partial queries. The redundancy in the query specification potentially breaches such integrity. As we seek to provide a natural view of data, this redundancy may inherently appear in the schema. For instance, the keyword area of authoring may be considered a full-fledged entity as well as an attribute subordinate to the entity *Reference*. If the user has specified *Reference*'s attribute *keyword*, the subsequent selection of an adverb phrase based on *Keyword* potentially introduces inconsistency. The model should allow the interface creator to express the redundancy of information in the schema so that the system may use it to avoid potential inconsistency in user queries.

6.2 Basic Concepts

Entities, *relationships*, and *relationship modifiers* describe the overall schema of a database. In this sense, the model may be called E-R-M model. Only entities are allowed to own *attributes*. Entities correspond to noun phrases (NPs) appearing as subjects, objects, and prepositional phrase (PP) objects. Relationships model domain-specific verbs, and take one or two entities as arguments. Relationship modifiers represent adverb phrases, such as *wh*-adverbs and prepositional phrases. Each relationship modifier takes two arguments: one for the base entity involved in modifying the relationship, and another for the relationship that it modifies. The arguments of both relationships and relationship modifiers can be specified by constraint formulas as well as atoms. In our model, a typical E-R relationship [16, 17] is represented by a relationship of fixed arity (≤ 2) and an arbitrary number of relationship modifiers.

Figure 6.1 shows a graphically represented schema. Rectangles, diamonds, and trapezoids represent entities, relationships, and relationship modifiers, respectively.

Note that our model avoids the need of introducing deep cases explicitly by assigning subjects and object to relationship arguments, and collecting the deep cases corresponding to adverb phrases as relationship modifiers.

6.2.1 Entities

Entities model not only objects with unique identity such as *Author* and *Book* but also mass nouns, such as *Salary,* if domain-specific verbs take them as subjects, objects, or PP objects. Count and mass entities have different *wh*-determiners in EnQL: "which" and "how much," respectively. Mass entities may have comparative adjectives as in "Who earn more salary than their managers?" An entity definition includes:

- a feature *countp*, which indicates the countability of an entity,
- a noun to be used for reference,
- a set $\mathcal{A}$ of attributes (or properties),
- a set $\mathcal{K}$ of key attributes ($\mathcal{K} \subseteq \mathcal{A}$),
- a set $\mathcal{P}$ of default projection attributes ($\mathcal{P} \subseteq \mathcal{A}$).

Entity Attributes An attribute is marked to indicate if it is qualified for a prenoun modifier. Key attributes are in general not allowed to appear as prenoun modifiers. All attributes may appear in postnoun modifier clauses. Each entity attribute refers to a *domain* and has a noun for its reference. A domain definition contains information on guiding the user's value creation, such as the type of pop-up menus.

6.2.2 Relationships

Relationship arguments are assigned their roles: subject or object. In the graphical schema representation, an arrow from an entity to a relationship indicates that the entity plays the subject role, while an arrow from a relationship to an entity indicates that the entity plays the object role. A relationship definition additionally includes:

- a feature *tense* to specify the legitimate tenses of a relationship,
- a verb to be used for reference.

Example The query Q1 illustrates the realization of two binary relationships: *Author_Book* ($Author_{subj}$, $Authored\_Book_{obj}$) and *Publish_Book* ($Publisher_{subj}$, $Book_{obj}$) underly the verbs "wrote" and "published," respectively.

The unary relationship *Receive_PhD* ($Author_{subj}$) with *tense* = *past* models the fact that some authors received PhD. When this relationship is realized in a query, an NP referring to *Author* appears in the subject position of verb "received PhD."

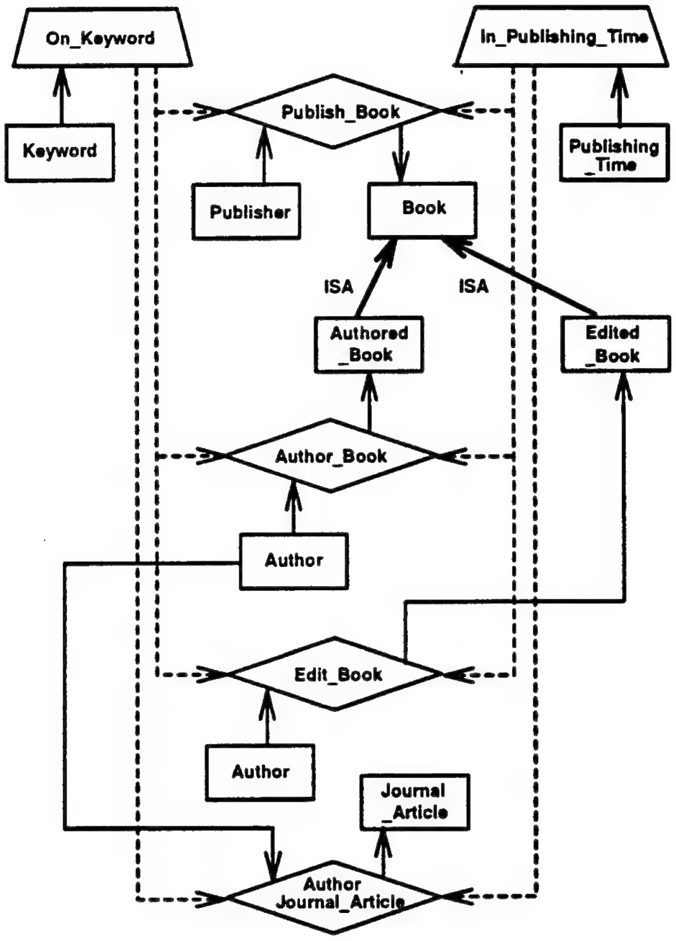


Figure 6.1: A Graphical EnQL Schema

While three-place relationships are conceivable to model bitransitive verbs (e.g., "x pays y \$$z$"), such relationships are substituted by two-place relationships by moving indirect objects to adverb positions ("x pays \$$z$ to y").

6.2.3 Relationship Modifiers

An arbitrary number of relationship modifiers may be associated with each relationship, and vice versa. As a result, the relationship argument of a relationship modifier is typically specified by a disjunction of relationships. For example, in Figure 6.1, *In_Publishing_Time* and *On_Keyword* modify the relationships *Author_Book*, *Edit_Book*, *Publish_Book*, and *Author_Journal_Article*. The relationship argument of these modifiers is then expressed by a disjunctive formula:

$$\textit{Author\_Book} \vee \textit{Edit\_Book} \vee \textit{Publish\_Book} \vee \textit{Author\_Journal\_Article}.$$

A relationship modifier may be realized with multiple prepositions. For example, although our convention affixes a representative preposition "In" to the base entity name "Publishing_Time," *In_Publishing_Time* may be realized not only as "in 1982" but also as "since 1982" or "before 1982."

Some relationship modifiers are shared by a set of relationships in the sense that two verbs sharing an NP also share adverb phrases. The query Q1 exemplifies this. If either of two verbs "wrote" and "published" is restricted by the adverb phrase "since 1982," the other is also restricted by the same adverb phrase. This information is useful for checking the semantic consistency of two related verb phrases. In the graphical schema representation, arrowed lines connect relationship modifiers to relationships. If the line ends with multiple relationships, they share the relationship modifier.

6.3 ISA Hierarchies

ISA hierarchies organize schema concepts by similarity and difference. Figure 6.1 also shows *ISA* relationships between entities. The semantics of the entity hierarchy is that if E_d is a descendent of E_a ($ISA(E_d, E_a)$), then E_d is a subset of E_a. E_d

inherits all attributes of E_a, and may define new attributes. The model imposes a mandatory rule regarding entity specialization:

> *If a set of relationships disjointly divides an entity set, create specialized entity sets, one for each of the relationships.*

For example, two relationships *Author_Book* and *Edit_Book* disjointly divide the entity set *Book* because a book is either authored or edited but not both. (An edited book, however, may contain many authored chapters or articles.) By the mandatory rule, two entities *Authored_Book* and *Edited_Book* are created as specializations of *Book*, and used for specifying the object arguments of the relationships. This mandatory entity specialization avoids nonsensical queries such as "Who wrote books edited by ..." Here, the entity *Authored_Book* referred to by "books" cannot be an argument of *Edit_Book*. Thus "edited by" is pruned from the choice set presented after the user's selection of "books."

The entity hierarchy enables users to query a specialized entity set. Let N_a and N_d be NPs realizing entities E_a and E_d. Users may ask:

"Which N_a are N_d?"

("Which theses are PhD theses?")

Note that reversing the order of N_a and N_d leads to trivial questions such as "Which PhD theses are theses?" Therefore EnQL does not support this type of query.

Relationships are also organized into hierarchies as shown in Figure 6.2. The arguments of a parent relationship subsume the arguments of all of its child relationships. A child relationship inherits all the modifiers associated with its parent. Additional relationship modifiers may be defined for the child relationship. The existence of additional relationship modifiers mandates relationship specialization. For instance, the relationship *Author_Journal_Article* is specialized from *Author_Reference* because the modifier *In_Journal* is applicable only to *Author_Journal_Article*. A relationship is also specialized without introducing new modifiers when its arguments are specialized. For example, *Author_Thesis* has specializations *Author_PhDThesis* and *Author_MasterThesis*.

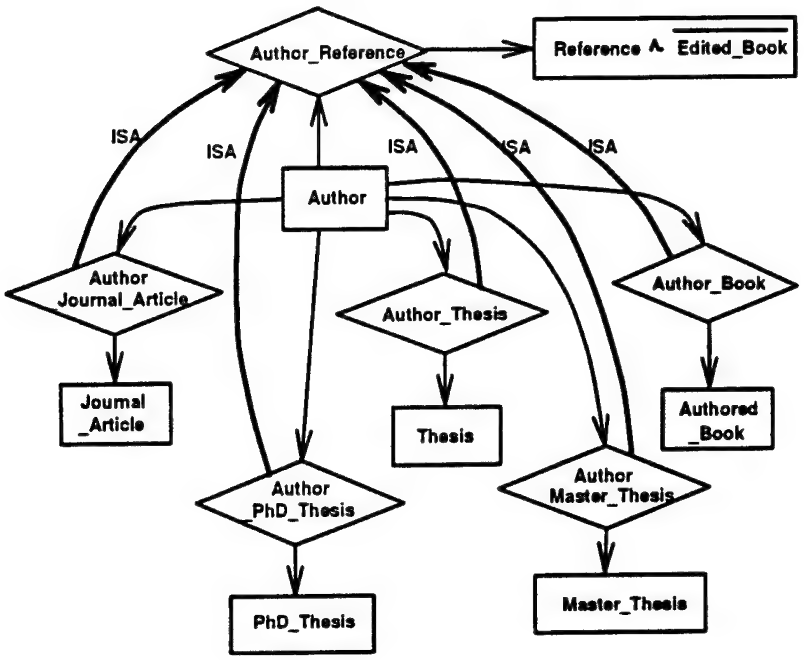


Figure 6.2: A Relationship *ISA* Hierarchy

Benefits The *ISA* hierarchies form a basis for:

- Extending unification in such a way that two atoms in *ISA* relationship unify to the specialized one. As a result, *Book* and *Authored_Book* unify to *Authored_Book*.
- Overloading attribute-based choices. For example, the choice "edition" is specified as the attribute of the entity *Book*, but also serves as an attribute of *Authored_Book* and *Edited_Book*, thus reducing the number of choices on the screen.
- Supporting the user's choice of commitment degree: Let $\mathcal{V}_x$ denote a set of verbs that can be attached to an NP referring to the entity E_x. Then $\mathcal{V}_d \subseteq \mathcal{V}_a$ holds for *ISA*(E_d, E_a). Similarly, let $\mathcal{A}_x$ be a set of attributes that can be matched by the entity specification E_x. Then, $\mathcal{A}_d \subseteq \mathcal{A}_a$ holds as well.
- Presenting general/specialized terms hierarchically on the menu.

- Organizing lexicon entries hierarchically such that the failure of unification at a nonleaf node guarantees the failure at all of its descendents. For example, if the match fails at *Books*, it is unnecessary to try to match *Authored_Book* and *Edited_Book*.

6.4 I-OVERLAP

Often two entity sets such as *Thesis* and *Technical_Report* overlap, even if they are not in *ISA* relationship. The *I-OVERLAP* relationship captures such intrinsically overlapping entity sets. This relationship provides the basis of determining legitimate noun qualifiers, such as "thesis" in "thesis technical reports," and qualified NP complements for establishing the entity identity as in "Which technical reports are 'Stanford' theses?" *I-OVERLAP* has following properties:

Symmetry:
$I\text{-}OVERLAP(E_1, E_2) \Rightarrow I\text{-}OVERLAP(E_2, E_1)$. Thus, if "thesis technical reports" is legitimate, so is "technical report theses."

Pseudotransitivity:
$ISA(E_1, E_2) \land I\text{-}OVERLAP(E_2, E_3) \Rightarrow I\text{-}OVERLAP(E_1, E_3)$. As a result, "PhD thesis technical reports" is also a legitimate NP.

With the *I-OVERLAP* relationship, entity sets with multiple parents are not necessary. As a result, the *ISA* hierarchies in our model retain the simplicity of tree structures. Compared with the lattice-based multiple inheritance, our inheritance model reduces the number of entity sets to represent in the schema significantly.

6.5 Derived Attributes and Subordinate Entities

It is desirable for entities, relationships, and relationship modifiers to be defined without redundancy. For example, if *Keyword* is modeled as the base entity of the relationship modifier *On_Keyword*, the keyword information does not appear in the definition of *Book*. However, to support the shorthand expression "DATABASE books," it is desirable to treat *Keyword* as if it were an attribute of *Book*. In Kaleidoscope, this is done by *derived attributes*: the attributes from a relationship modifier's base entity are imported to the argument entity of the relationship.

Similarly, it is desirable to refer to some entities as if they were subordinate to others, as in "which publisher's books" or "which book's publishers." The relationship *Publish_Book* is implicit in both cases. The schema may define an argument entity of a binary relationship subordinate to the other.

6.6 Internal Query Language

This section defines an internal query language (IQL) for representing the query meaning and integrity constraints, and presents a mapping from our model to the relational model.

Let R, M, and S be the sets of symbols representing relationships, relationship modifiers, and built-in predicates. S includes $=$, $\neq$, $>$, $<$, $\geq$, $\leq$, *between*, and *not between* as its members. Note that S is closed under negation. A query may contain range-restricted variables for entity sets and relationships.

6.6.1 Query Meaning Representation

A query Q is a conjunction of positive literals P_i:

$$Q = \{(e_1, e_2, \ldots) \mid \bigwedge_{i=1}^{n} P_i\}$$

where $e_1, e_2, \ldots$ are free entity variables. Let p_i be the predicate symbol of P_i, then p_i is drawn from R, M, or S. All relationship variables, and the entity variables which do not appear as free variables are existentially quantified. The following condition holds for Q: for each literal P_i, there exists at least one literal $P_j (i \neq j)$ such that the variables in the arguments of P_i and P_j overlap.

EnQL grammar encodes the following mapping from EnQL to IQL:

- For each reference to an entity set or an individual entity in EnQL, an entity variable is created. If an entity set reference is qualified by a *wh*-word, the variable is free; otherwise, it is existentially quantified.
- If $p_i \in R$, P_i has three arguments: a relationship variable, and variables for the subject and object entities. If p_i represents a unary relationship, one argument is left empty.
- If $p_i \in M$, the first argument of P_i is a relationship variable, and the second is a variable for the base entity of p_i.
- If $p_i \in S$, the first argument of P_i is a pair of an entity variable and an attribute. Either constants or pairs of entity variables and attributes are qualified for the remaining arguments.

Example The system builds the following conjunctive query incrementally while the user creates the query Q1.

$$\begin{gathered}\{(x,y) \mid (\mathit{Author\_Book}\ r_1\ x\ y) \wedge (=\ y.\mathit{keyword}\ \text{“DATABASE”}) \wedge \\ (\mathit{Publish\_Book}\ r_2\ p\ y) \wedge (=\ p.\mathit{name}\ \text{“McGraw-Hill”}) \wedge \\ (\mathit{In\_Publishing\_Time}\ r_1\ t) \wedge (\geq\ t.\mathit{year}\ 1982)\ \} \\ \text{where } x \in \mathit{Author},\ y \in \mathit{Authored\_Book}\ p \in \mathit{Publisher}, \\ t \in \mathit{Publishing\_Time}\ r_1 \in \mathit{Author\_Book}, r_2 \in \mathit{Publish\_Book}.\end{gathered}$$

In this query, variables p, t, r_1, r_2 are existentially quantified.

6.6.2 Integrity Constraints for Intraquery Cooperation

An integrity constraint is a negated conjunction of literals:

$$\neg\, (\bigwedge_{i=1}^{m} L_i)$$

where the predicate symbol l_i of L_i is drawn from R, M, or S. We restrict L_i to be positive if l_i belongs to R and M. With S closed under negation, the literals based on the symbols in S could be treated as either positive or negative.

To illustrate, the integrity constraint IC1 is formally expressed as:

$$\neg\,((\mathit{Teach}\ r\ i\ c) \wedge (\mathit{I\text{-}OVERLAP}\ i\ s) \wedge s \in \mathit{Student}$$
$$\wedge\,(=\ c.\mathit{dept}\ \text{"CS"}) \wedge (\geq\ c.\mathit{number}\ 400))$$

Here, for the simplicity of presentation, we left out the declaration of range variables except for s. If all literals of an integrity constraint are true in a query, the system warns the user of the integrity constraint violation. For this inference, Kaleidoscope uses OPS5 [25]. Thus, Kaleidoscope represents the above integrity constraint by a production rule:

```
IF     (Teach r i c) ∧ (I-OVERLAP i s) ∧ s ∈ Student ∧
       (= c.dept |CS|) ∧ (≥ c.number 400)
THEN   MAKE (Warning
       |Student instructors never teach CS400 or higher level courses|).
```

The Integrity constraint IC2, on the other hand, has a different type of **THEN** part to derive literals:

```
IF     (Teach r i c) ∧ (I-OVERLAP i s) ∧
       (= c.dept |CS|) ∧ (≥ c.number 400)
THEN   MAKE s ∉ Student
```

In the actual OPS5 implementation of integrity constraint rules, all literals are defined with two system-defined arguments:

1. The classification (0: user-selected, 1: trivial, 2: informative, 3:critical).

2. The time stamp of creation.

The first is needed for the application of the information presentation heuristics presented in Chapter 2. The latter is useful for removing all user-selected and derived literals when the user retracts selections.

6.7 Mapping to Relational Storage

6.7.1 Mapping

A mapping from an EnQL schema to a relational schema consists of:

Entity Name	*Attribute*	*Range Relation*	*Range Field*	*Joins*
Reference	**id**	reference	id	
	title	reference	title	
	published-year	reference	year	
	keyword	keyword_view	string	3, 4
		reference_keyword		
Book	**id**	book	id	
	title	reference	title	5
	published-year	reference	year	5
	keyword	keyword_view	string	3, 6
	edition	book	edition	
	length	book	length	
		reference_keyword		
Authored_Book	**id**	book	id	7
	title	reference	title	5
	published-year	reference	year	5
	keyword	keyword_view	string	3, 6
	edition	book	edition	
	length	book	length	
		reference_keyword		
		author_reference		

Table 6.1: Entity-level Mapping of *Reference/Book/Authored_Book*

- An attribute-level mapping, which maps entity attributes to database (DB) relation attributes.
- An entity-level mapping, which adds restrictions and joins to the collection of attribute-level mappings.
- A relationship-level mapping, which defines joins between the arguments of a relationship, and between the argument of a relationship and the base entities of its modifiers.

Restrictions may add at entity and relationship levels, but they are trivial to process in query translation. Tables 6.1 and 6.2 show the entity-level mapping and a reference join table, respectively. Entity keys are printed in boldface. We have shown

Id	Relation 1	Field 1	Pred	Relation 2	Field 2
1	person_name	id	=	person_organization	person
2	author_reference	author	=	person_name	id
3	keyword_view	id	=	reference_keyword	keyword
4	reference	id	=	reference_keyword	reference
5	book	id	=	reference	id
6	book	id	=	reference_keyword	reference
7	author_reference	reference	=	book	id
8	book	id	=	publish_reference	reference
9	organization	id	=	publish_reference	organization

Table 6.2: Join Table

a corresponding relational storage schema based on the structural data model [75] in Figure 3.2. Table 6.3 shows the relationship-level mapping. At this level, multiple entity sets may map to a single relation. The appearance of identical relations on the left and right of the predicate in the rows of the table 6.3 indicates such sharing of relations.

6.7.2 Query Translation

Query translation proceeds as follows:

1. From an IQL representation, create an input record with:
    - Entity variables and their types.
    - Free entity variables and projection attributes: the latter are acquired either by prompting the user with pop-up menus or by retrieving the default set of attributes.
    - Entity restrictions and joins.
    - Relationships and their modifiers: collect relationship modifiers within the relationship that they are associated with. Relationship variables are removed in this process.

Relationship			
Entity 1	*relation1.field1*	*relation2.field2*	*Entity 2/Modifier*
Author_Reference			
Reference	keyword_view.id	keyword_view.id	*In_Keyword*
Reference	reference.id	reference.id	*In_Publishing_Time*
Author	author_reference.reference	reference.id	*Reference*
Reference	reference_keyword.keyword	reference_keyword.keyword	*In_Keyword*
Author_Book			
Author	author_reference.reference	book.id	*Authored_Book*
Authored_Book	keyword_view.id	keyword_view.id	*In_Keyword*
Authored_Book	reference.id	reference.id	*In_Publishing_Time*
Author	author_reference.reference	author_reference.reference	*Authored_Book*
Authored_Book	reference_keyword.keyword	reference_keyword.keyword	*In_Keyword*

Table 6.3: An Exemplary Relationship-Level Mapping

2. Create a hash table for keeping track of entity views. An entity view is created for each entity variable in the query and contains a minimal list of attributes: key attributes, projection and restriction attributes, and join attributes that are either explicit in the user query or required by the entity and relationship-level mapping.

3. For each relationship R with a list of modifiers $\{M_i\}$, do the following:

   (a) Collect entity views corresponding to the entity variables found in R and M_i's.

   (b) Create DB joins across these entity views as defined by the relationship-level mapping. Also unify multiple instantiations of identical DB relations in this process.

4. For each instance of *I-OVERLAP* in the query, create an equijoin of two entity views.

5. Collect the DB projection attributes, DB relation instances (pairs of relation names and unique identifiers), DB table joins, and DB table restrictions, and create an SQL query.

6.8 Summary

This chapter has presented a data model for supporting EnQL queries. Its features are:

- Entities, relationships, and relationship modifiers describe the overall structure of a schema. Unary and binary relationships model the semantics of intransitive and transitive verbs, respectively. Relationship modifiers are intended to support full-fledged adverb phrases involving complex noun phrases.
- Generalization hierarchies form a basis of structuring schema concepts and menu choices. To users, hierarchically organized choices provide a range of commitment in choosing references.
- The notions of subordinate entities and derived attributes enable users to create shorthand expressions for referring to relationships and relationship modifiers.
- The notion of intrinsically overlapping entity sets provides a semantic basis of generating noun modifiers and establishing identity of entities belonging to different sets. In addition, it avoids the need for the explicit creation of entity sets with multiple ancestors, thus reducing the number of entity sets to represent in the schema.

We have also specified a query language IQL for the internal meaning representation of EnQL queries and expression of integrity constraint rules. Finally, we have defined a mapping from the EnQL model to the relational model and a query translation procedure.

Chapter 7

Grammar Formalism for Choice Generation

To provide a flexible grammar formalism for choice generation, Kaleidoscope extends Context-Free Grammar (CFG) in two ways. First, it augments each category symbol with a list of *feature attributes.* The role of these feature attributes is similar to attributes [23], registers [78], and features [59] of other grammar formalisms in providing context-dependency. Second, Kaleidoscope's formalism provides several types of *procedural decoration* for attachment to grammar rules. This decoration enables the interface designer to capture arbitrary context-dependent constraints and actions in grammar.

7.1 Augmented Context-Free Grammar

A CFG defines a set of rewrite rules over category and terminal symbols. Each rule rewrites the symbol on the left-hand side (LHS) to a sequence of one or more right-hand side (RHS) symbols. The grammar designates a special symbol as the start symbol. We choose the symbol S to denote this start symbol. The set of rules rewriting preterminal symbols to terminal symbols is called *lexicon* and maintained separately from the rest of rewrite rules.

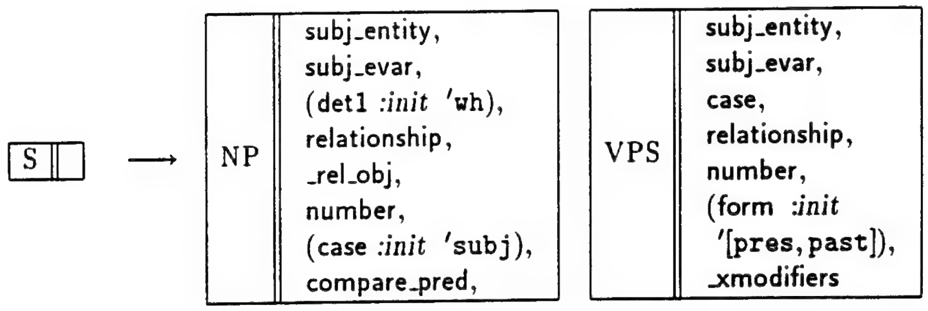


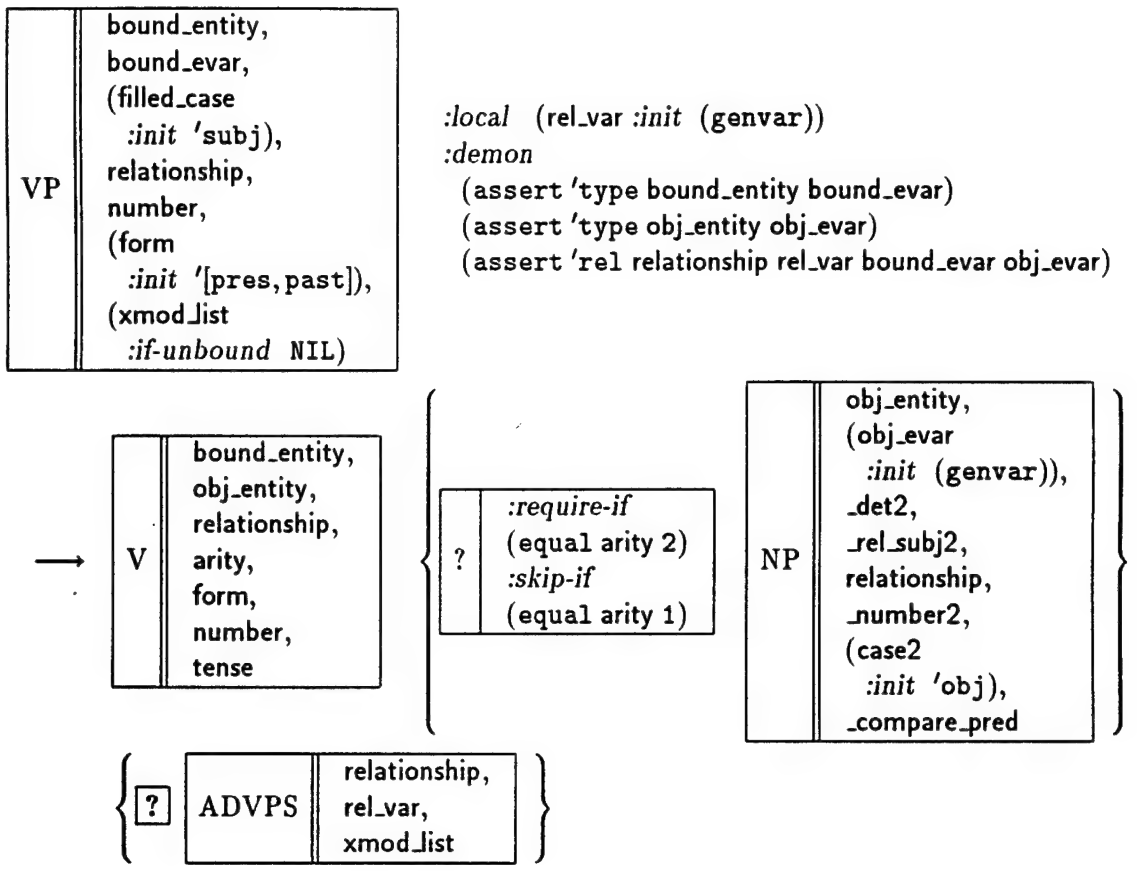
Figure 7.1: A Top-Level Grammar Rule

7.1.1 Feature Attributes

Kaleidoscope augments each category symbol with an ordered list of feature attributes to formalize the context of a constituent category. A feature attribute is called semantic if its domain of values depends on schema terms. Otherwise, it is called syntactic. With feature augmentation, grammar rules rewrite in terms of a category symbol and a list of variables representing feature attributes. We call these variables *feature variables*. The scope of a feature variable is a single rule.

Figure 7.1 shows a top-level EnQL grammar rule. This rule states that a query (S) is made up of a noun phrase (NP) and a conjunction of verb phrases (VPS). A collection of feature variables (typed in a sans serif style) appear boxed next to the category symbol. Figure 7.2 shows a more complex rule on VP for constructing a single verb phrase in VPS. This rule states that a verb phrase (VP) comprises a verb of present or past form, optionally followed by an object noun phrase (NP) and a list of adverb phrases (ADVPS). The symbols inside { $\boxed{?}$ ... } form an optional sequence. Those immediately following $\boxed{?}$ make conditions for turning the sequence into a mandatory or null sequence. This feature is discussed further in Section 7.4.

The run-time binding of most feature variables come from lexicon. However, some variables representing syntactic features are often initialized by grammar rules. The rule in Figure 7.1 illustrates such initialization. Initializing detl to wh mandates the first NP of a query to begin with a *wh*-word. The rule also assigns the constant



```
VP || bound_entity,
      bound_evar,
      (filled_case
        :init 'subj),
      relationship,
      number,
      (form
        :init '[pres,past]),
      (xmod_list
        :if-unbound NIL)

:local (rel_var :init (genvar))
:demon
  (assert 'type bound_entity bound_evar)
  (assert 'type obj_entity obj_evar)
  (assert 'rel relationship rel_var bound_evar obj_evar)

→ V || bound_entity,
       obj_entity,
       relationship,
       arity,
       form,
       number,
       tense

  { ? | :require-if
        (equal arity 2)
        :skip-if
        (equal arity 1)
    NP || obj_entity,
          (obj_evar
            :init (genvar)),
          _det2,
          _rel_subj2,
          relationship,
          _number2,
          (case2
            :init 'obj),
          _compare_pred }

  { ? ADVPS || relationship,
               rel_var,
               xmod_list }
```

Figure 7.2: A Grammar Rule on VP

subj to the feature variable case to indicate that the NP plays the subject role in the sentence. The feature variable form in VPS is also initialized by the rule to limit the verb phrases to present and past forms. Kaleidoscope provides two procedural decorations on feature variables, one for initialization and another for specification of default values. The semantics of these decorations is further discussed in Section 7.4.

Unification of feature variable bindings in rule rewriting blocks unnecessary application of child rules. For example, the appearance of subj_entity and relationship in both NP and VPS requires that the bindings of these variables at two positions unify. If they don't, rules on VPS will not be activated. A special unification procedure may be specified for feature attributes. For example, most semantic features are associated with the unification procedure that recognizes *ISA* hierarchies. The default unification procedure does not.

7.1.2 Partial Value Representation

Partial values express constraints on the ultimate binding of variables. For example, the initial value '[pres,past] of form in Figure 7.1 specifies that the value of form is either present or past. Partial values enable compact expression of constraints on feature attributes, thus reducing the size of grammar and lexicon. The run-time efficiency of choice generation also increases as a result. Kaleidoscope provides a limited language for partial value representation.

A *disjunction formula* expresses a constraint that the binding of a variable should be restricted to a set of constants. Such a set is denoted by a list of constants enclosed with a pair of brackets.

$$(\mathsf{x} = [c_1, c_2, \ldots, c_n]) \;\equiv\; (\mathsf{x} = c_1 \vee \mathsf{x} = c_2 \vee \ldots \vee \mathsf{x} = c_n).$$

A *negation formula* expresses a constraint that a variable should not be bound to any element of a given list. For the sake of notational consistency with disjunction formulas, we represent a negation formula in terms of a negated disjunction formula:

$$\begin{aligned}(\mathsf{x} = \overline{[c_1, c_2, \ldots, c_n]}) \;&\equiv\; (\overline{\mathsf{x} = c_1} \wedge \overline{\mathsf{x} = c_2} \wedge \ldots \wedge \overline{\mathsf{x} = c_n}) \\ &\equiv\; (\overline{\mathsf{x} = c_1 \vee \mathsf{x} = c_2 \ldots \vee \mathsf{x} = c_n}).\end{aligned}$$

Partial values for semantic feature attributes may contain the nonleaf nodes of *ISA* hierarchies. Let c be such a constant. If c appears in a disjunction formula for the variable x, x may be bound to not only c but also any descendent of c. If c appears in a negation formula for x, c and its descendents cannot be taken as x's value. For example, a disjunction formula entity = [*Thesis*, *TechReport*] states that **entity** can be bound to *PhDThesis* and *MasterThesis* as well as *Thesis* and *TechReport*.

We say that a disjunctive value formula or a negation formula is *minimized* if no constant in the formula is repeated or in *ISA* relationship to another. In Kaleidoscope, the binding of a feature variable takes one of the following forms:

- A constant.
- A minimized disjunction formula.
- A minimized negation formula.
- A pair of minimized disjunction and negation formulas in conjunction.

7.1.3 Feature Binding Unification

Unification is an operation of merging two partial object descriptions [39]. A feature binding list F combined with a category C forms a partial description of constituents in C. Let F_1 and F_2 be bindings on two feature attribute sets A_1 and A_2, respectively. F_1 and F_2 unify if $f_1^{a_j}$ and $f_2^{a_j}$ unify for any attribute $a_j \in A_1 \cap A_2$, where $f_i^{a_j}$ represents the binding of the attribute a_j appearing in F_i ($i = 1, 2$). Let $f_1^{a_j} \oplus f_2^{a_j}$ denote the result of successful unification of $f_1^{a_j}$ and $f_2^{a_j}$. Then the result of successful unification of F_1 and F_2 is:

$$F_1 \oplus F_2 \;=\; \bigcup \left(\begin{array}{c} \cup_{a_i \in A_1, a_i \notin A_2} f_1^{a_i} \\ \cup_{a_j \in (A_1 \cap A_2)} (f_1^{a_j} \oplus f_2^{a_j}) \\ \cup_{a_k \in A_2, a_k \notin A_1} f_2^{a_k} \end{array} \right)$$

Below we present the operational semantics of unifying $f_i^{a_j}$'s based on the set-oriented view of partial values. We say that a formula f has a ***positive constant set*** (PSET) P if P is the set of disjunctive constants in f. Similarly, we say that f has a *negative constant set* (NSET) N if N is the set of negative constants in f's negated formula. A constant value may be regarded as a PSET of cardinality 1.

Based on these set terminologies, unification of two formulas f_1 (which has P_1 and N_1 as PSET and NSET, respectively) and f_2 (which has P_2 and N_2 as PSET and NSET, respectively) is specified as follows:

1. *Merging PSETs:* If two PSETs P_1 and P_2 are not null, compute their intersection by the following semantics:

   - collect p_{1i} (or p_{2j}) if $p_{1i} = p_{2j}$ where $p_{1i} \in P_1$ and $p_{2j} \in P_2$,
   - collect p_{1i} if $ISA(p_{1i}, p_{2j})$ holds, or p_{2j} if $ISA(p_{2j}, p_{1i})$ holds.

   In the absence of *ISA* information, this corresponds to usual set intersection. If any of P_1 and P_2 is null, take non-null one. If both of them are null, return NIL.

2. *Merging NSETs:* Make union of two NSETs N_1 and N_2. Eliminate any item that is a specialization of another in this process.

3. *Subtracting NSET from PSET:* Remove an item p from the collected PSET if p or p's generalization appears in NSET, or if a set of p's specializations covering p appear in NSET.

4. *Removal of Tautology in NSET if PSET is not null:* Remove tautological constants from NSET. A constant n in NSET is tautological if no constant in PSET is a generalization of n.

Example We illustrate the feature binding unification over the noun category. The following table presents sample entries in this category with their entity bindings. The rightmost column shows the result of unifying these entries with an input feature constraint {entity $=$ *Reference* $\wedge$ $\overline{\textit{Edited\_Book}}$}.

Choice String	Choice Binding	Unification Result
"authors"	entity = *Author*	*FAIL!*
"references"	entity = *Reference*	entity = *Reference* $\wedge$ $\overline{\textit{Edited\_Book}}$
"theses"	entity = *Thesis*	entity = *Thesis*
"PhD theses"	entity = *PhDThesis*	entity = *PhDThesis*
"technical reports"	entity = *TechReport*	entity = *TechReport*
"books"	entity = *Book*	entity = *Book* $\wedge$ $\overline{\textit{Edited\_Book}}$
		$\equiv$ *Authored_Book*

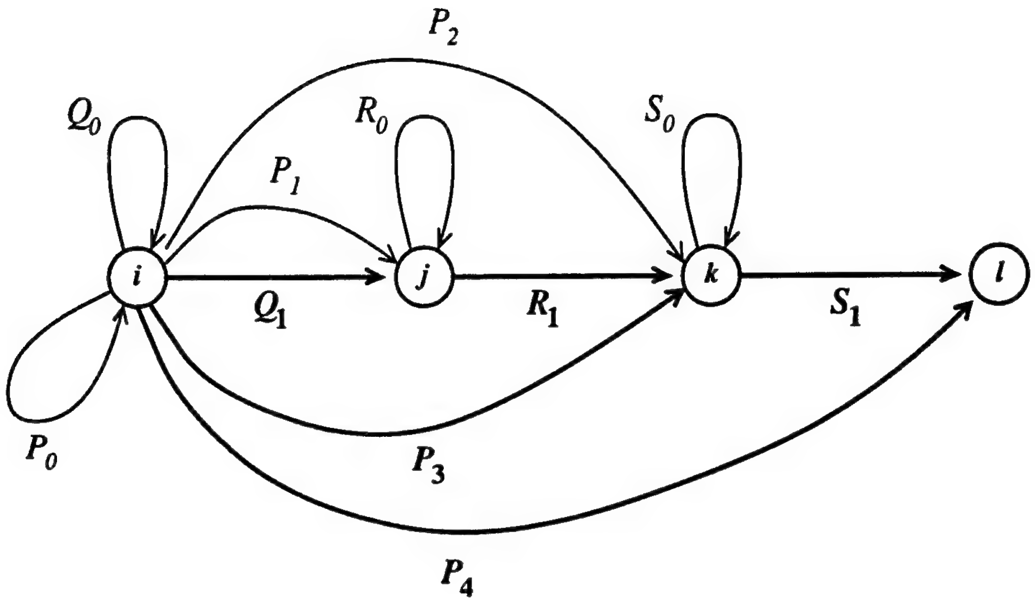
This process actually occurs after choosing the choice "wrote" in the course of constructing the query Q1.

7.2 Partial Query Representation

A partial query state consists of a collection of well-formed constituent structures and pending hypotheses on completing the partial query. A proper representation of this state is prerequisite for the system's exhibition of coherent behavior to the user. Kaleidoscope employs a run-time data structure called ***chart*** for the partial query representation.

Commonly used for syntactic analysis in natural language processing [36], a chart consists of a sequence of vertices and a set of edges connecting vertices. In Kaleidoscope, a vertex represents the point of user-system interaction. Each edge represents the state of a grammar rule applied to the partial query, and is labeled with the rule's LHS symbol. Two types of edges are distinguished: ***active*** and ***inactive*** edges. An inactive edge represents a completed constituent structure, and corresponds to a node in the parse tree. An active edge represents a partially completed or just activated grammar rule awaiting the completion of constituents.

A chart grows monotonically by incorporating new vertices and edges. Edges are never deleted or modified except by the user's explicit request to retract previous selections. As the user constructs a query, alternative parse trees are constructed in parallel. This parallel monotonic nature of the chart also makes it suitable for keeping track of feature bindings.



Rule: P $\rightarrow$ Q R {$\boxed{?}$ S}

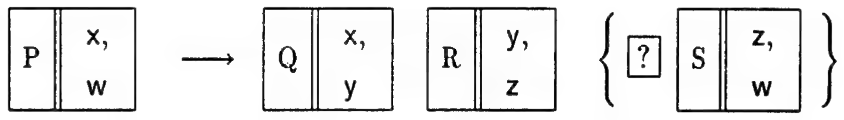
Order of Edge Instantiation: P_0 Q_0 | $\boldsymbol{Q_1}$ P_1 R_0 | $\boldsymbol{R_1}$ $\overset{P_2}{\boldsymbol{P_3}}$ S_0 | $\boldsymbol{S_1}$ $\boldsymbol{P_4}$

Edge Content

Edge	State	Predecessor	Subsumed	Outstanding	Remaining Symbols
P_0	active			Q_0	R (? S)
P_1	active	P_0	$\boldsymbol{Q_1}$	R_0	(? S)
P_2	active	P_1	$\boldsymbol{Q_1}$ $\boldsymbol{R_1}$	S_0	
$\boldsymbol{P_3}$	inactive	P_1	$\boldsymbol{Q_1}$ $\boldsymbol{R_1}$		
$\boldsymbol{P_4}$	inactive	P_2	$\boldsymbol{Q_1}$ $\boldsymbol{R_1}$ $\boldsymbol{S_1}$		

Figure 7.3: Example of Edge Creation on Chart

Figure 7.3 shows an exemplary chart created for the hypothetical grammar rule:



Bold edges represent inactive edges, and regular edges represent active edges awaiting extension. In Section 7.5, we describe the control flow of constructing the chart.

7.3 Lexicon

In Kaleidoscope, lexicon supplies candidate menu choices to the choice generator. Lexicon entries satisfying the context-sensitive constraints of grammar are presented on the menu. For the user, lexicon provides an additional layer of database view. Removal of entries from the lexicon disables reference to corresponding schema terms.

7.3.1 Preterminal Category Definition

Kaleidoscope's lexicon is organized by preterminal categories. Table 7.1 shows the definition of a preterminal category ENTITY-SET-N and sample entries. A preterminal category defines:

- Feature attributes.
- Special unify functions for feature attributes.
- A menu window.
- A collection of lexicon entries, each of which carries a feature binding list. Lexicon entries may specify a demon function to be executed upon the user's selection. Demon functions are executed with two arguments: the string and feature binding list of the selected choice.
- An optional *pivot feature,* which provides the basis of organizing lexicon entries hierarchically. For example, the pivot feature specification of ENTITY-SET-N organizes the nouns representing entity sets by entity *ISA* hierarchies.

ENTITY-SET-N:	
menu window:	Concept-Noun
feature attrs:	entity, v_subj, v_obj, countp, number
unify fn:	isa-unify: entity, v_subj, v_obj
pivot feature:	entity
"references"	
entity =	*Reference*
v_subj =	NIL
v_obj =	[*Author_Reference*, *Edit_Book*, *Publish_Book*, ...]
countp =	plus
number =	pl
"books"	
entity =	*Book*
v_subj =	NIL
v_obj =	[*Author_Book*, *Edit_Book*, *Publish_Book*]
countp =	plus
number =	pl
"(authored) books"	
entity =	*Authored_Book*
v_subj =	NIL
v_obj =	*Author_Book*
countp =	plus
number =	pl

Table 7.1: Category ENTITY-SET-N and Sample Entries

FINISH:	
menu window:	Finish-Phrase
feature attrs:	root-cat, start-vertex-id
before fn:	finish_before_fn
" "	*demon fn:* finish_demon_fn

Table 7.2: Category FINISH and Its Sole Entry

Model Concept	Category	Exemplary Choices
Entity	ENTITY-SET-N	PhD Theses, authors
	WH-PN	who, whom, whose
	INDIV-N	<PhD Thesis>, <author>
Attribute	ATTR-N-BE	name is, id is, keyword is
	ATTR-DEMON	<name>, <id>, <keyword>
	PRENOM-DEMON	<keyword>
Relationship	V (form = past)	wrote,
	(form = ppby)	written by
	(form = beppby)	were written by
Relationship Modifier	PREP	at, on
	WH-ADV	when, where
Possessive Entity	POSS-SET-N	publisher's
	POSS-INDIV-N	<publisher>'s
Subordinate Entity	POSSED-SET-N	books
Overlapping Entities	ENTITY-MOD	thesis, technical report

Table 7.3: Model Concepts and Preterminal Categories

- An optional *before function,* which is applied to matched lexicon entries before being presented to the user. This function is useful for computing choices dynamically from the partial query state. For example, the choice for parenthesizing Q1's complex noun phrase is computed by the before function of the category FINISH. Table 7.2 shows the definition of this category and its sole entry. The function `finish_before_fn` follows up the partial parse tree searching the node whose category is the same as the value of **root-cat** and produces a choice string by concatenating the leaf nodes of the subtree.

7.3.2 From Model to Lexicon

Kaleidoscope's model-based approach generates lexicon automatically from schema terms and a collection of English terms. Appendix B lists the tabular representation of information needed for this automatic lexicon generation.

Table 7.3 presents sample preterminal categories and their source concepts of the

EnQL model. In addition to syntactic feature attributes such as **form**, each of these categories defines a *key* semantic feature attribute corresponding to its source model concept. For example, the feature definition of the category ENTITY-SET-N includes **entity** as its key feature attribute. Most categories define auxiliary semantic feature attributes derived from other related model concepts. The attributes **v_subj** and **v_obj** of the category ENTITY-SET-N are such features. These attributes specify the verbs that take the noun entry as the subject and object, respectively. Auxiliary semantic feature attributes are useful for pruning *dead-end choices*. When selected by the user, these choices lead the user to an empty menu state, thus forcing the user to backtrack previous selections. For example, unconstrained application of the S rule of Figure 7.1 presents the entity nouns with no follow-up verbs as choices for the first noun. The noun "keywords," which represents the entity *Keyword*, is such a noun. The user's selection of this choice for the subject of a sentence leads to an empty menu state because *Keyword* is only modeled as the base entity of *On_Keyword*. In the next section, we show how Kaleidoscope's preterminal category decoration prunes dead-end choices using auxiliary semantic feature attributes.

7.4 Procedural Decorations

The second CFG extension of Kaleidoscope captures the computation to carry out inside a rule body in parallel with rule rewriting. For this purpose, Kaleidoscope supports a few types of procedural decoration on feature variables, optional sequences, preterminals, and rules. Procedures in these decorations are arbitrary LISP s-expressions which may have feature variables as parameters.

The goal of procedural decoration is:

1. To compose the binding of one feature variable from the bindings of others in an arbitrary way.

2. To prune syntactically valid but semantically infeasible extension of the partial query.

3. To prune dead-end choices.

4. To interact with the nonlinguistic part of the system:

   (a) To construct the partial query meaning and pass it to the rule-based inference system.

   (b) To provide an escape mechanism for accessing desired semantic information missing in the lexicon.

A pure unification-based grammar formalism does not provide the power and flexibility of Kaleidoscope's procedural decorations. This section presents the syntax and semantics of procedural decoration types and justify their utility in choice generation.

7.4.1 Feature Variable Decorations

Feature variables may have two kinds of decoration governing their initial and default binding:

(x *:init* $\mathcal{E}$): If x appears on the left-hand side of the rule, evaluate $\mathcal{E}$ as the initial value to be unified with the corresponding value inherited from its parent rule. Otherwise, evaluate $\mathcal{E}$ if x is unbound whenever it is needed by the rule rewriting process or evaluating other procedural decorations.

(x *:if-unbound* $\mathcal{E}$): Evaluate $\mathcal{E}$ as its default value if x is unbound when needed for evaluating other procedural decorations and passing feature bindings up to parent rules. This decoration is not evaluated when passing feature bindings down to child rules while the *:init* decoration is.

The scope of a feature variable decoration is a single grammar rule. Thus the actual position of decoration in the rule does not matter. It is meaningless to attach *:if-unbound* decoration to a feature variable in the presence of *:init*.

We have shown the exemplary use of the *:init* decoration in Figure 7.1. The *:if-unbound* decoration is useful for specifying the default binding of unbound feature variables when they are needed by the parent rule.

7.4.2 Optional Sequence Decorations

The optional occurrence of symbols is often context-dependent. Two types of decorations may appear at the beginning of optional sequences:

:require-if $\mathcal{E}$: If $\mathcal{E}$ evaluates to non-NIL value, the optional sequence becomes a required sequence.

:skip-if $\mathcal{E}$: If $\mathcal{E}$ evaluates to non-NIL, the optional sequence should be skipped.

The example of using these decorations appears in the VP rule of Figure 7.2:

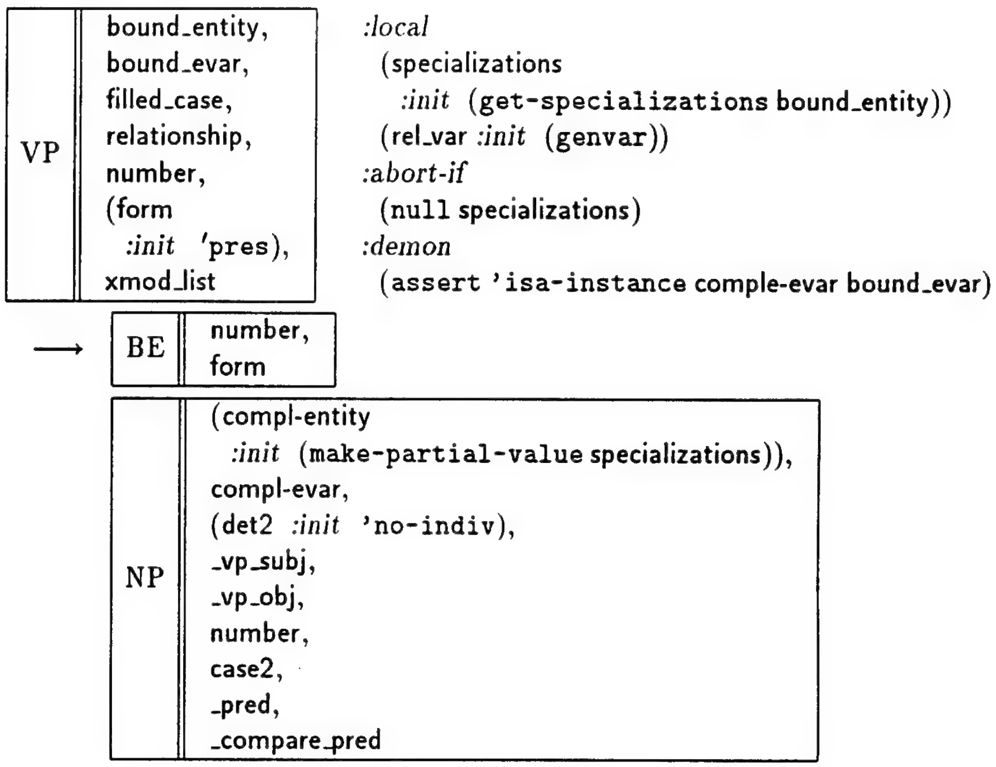
$$\left\{ \begin{array}{|c|l|} \hline ? & \begin{array}{l} \textit{:require-if} \\ \texttt{(equal arity 2)} \\ \textit{:skip-if} \\ \texttt{(equal arity 1)} \end{array} \\ \hline \end{array} \quad \begin{array}{|c||l|} \hline \text{NP} & \begin{array}{l} \textsf{obj\_entity,} \\ \textsf{(obj\_evar} \\ \quad \textit{:init}\ \texttt{(genvar)),} \\ \textsf{\_det2,} \\ \textsf{\_rel\_subj2,} \\ \textsf{relationship,} \\ \textsf{\_number2,} \\ \textsf{(case2} \\ \quad \textit{:init}\ \texttt{'obj),} \\ \textsf{\_compare\_pred} \end{array} \\ \hline \end{array} \right\}$$

This optional sequence becomes mandatory if the proceeding verb is transitive, namely, the arity of the relationship represented by the verb is 2. If the arity is 1, the choice generator skips the optional sequence.

7.4.3 Rule Decorations

Kaleidoscope supports four types of rule decoration for

- specifying a condition for invalidating grammar rule application (*:abort-if*).
- attaching arbitrary computation to be carried out inside the rule body (*:on-exit* and *:demon*).



```
VP || bound_entity,              :local
      bound_evar,                  (specializations
      filled_case,                   :init (get-specializations bound_entity))
      relationship,                (rel_var :init (genvar))
      number,                    :abort-if
      (form                        (null specializations)
        :init 'pres),            :demon
      xmod_list                    (assert 'isa-instance comple-evar bound_evar)
  ⟶ BE || number,
           form
     NP || (compl-entity
              :init (make-partial-value specializations)),
           compl-evar,
           (det2 :init 'no-indiv),
           _vp_subj,
           _vp_obj,
           number,
           case2,
           _pred,
           _compare_pred
```

Figure 7.4: Another Grammar Rule on VP

- declaring and initializing local variables, which do not represent any feature attributes (*:local*).

We present below the syntax and semantics of these decorations:

:abort-if $\mathcal{E}$: Abort the rule application if $\mathcal{E}$ evaluates to a non-NIL value.

:on-exit $\mathcal{E}_1, \mathcal{E}_2, \ldots, \mathcal{E}_n$: Evaluate $\mathcal{E}_1, \mathcal{E}_2, \ldots, \mathcal{E}_n$ in this order when the decorated rule is completed.

:demon $\mathcal{E}_1, \mathcal{E}_2, \ldots, \mathcal{E}_n$: Evaluate $\mathcal{E}_1, \mathcal{E}_2, \ldots, \mathcal{E}_n$ in this order whenever the list of constituents subsumed by the activated rule expands.

:local V_1, $V_2, \ldots$: Define a list of local variables in a rule body. Here V_i is a variable with or without *:init* or *:if-unbound* decorations.

In the chart representation of the partial query, the *creation* of new active edges (N), the *extension* of existing active edges (E), and the *inactivation* of active edges (I) form three major types of change in the state of an activated grammar rule. Thus we define the execution timing of procedural decorations by these three events. The following table summarizes the applicability of rule decorations by event types:

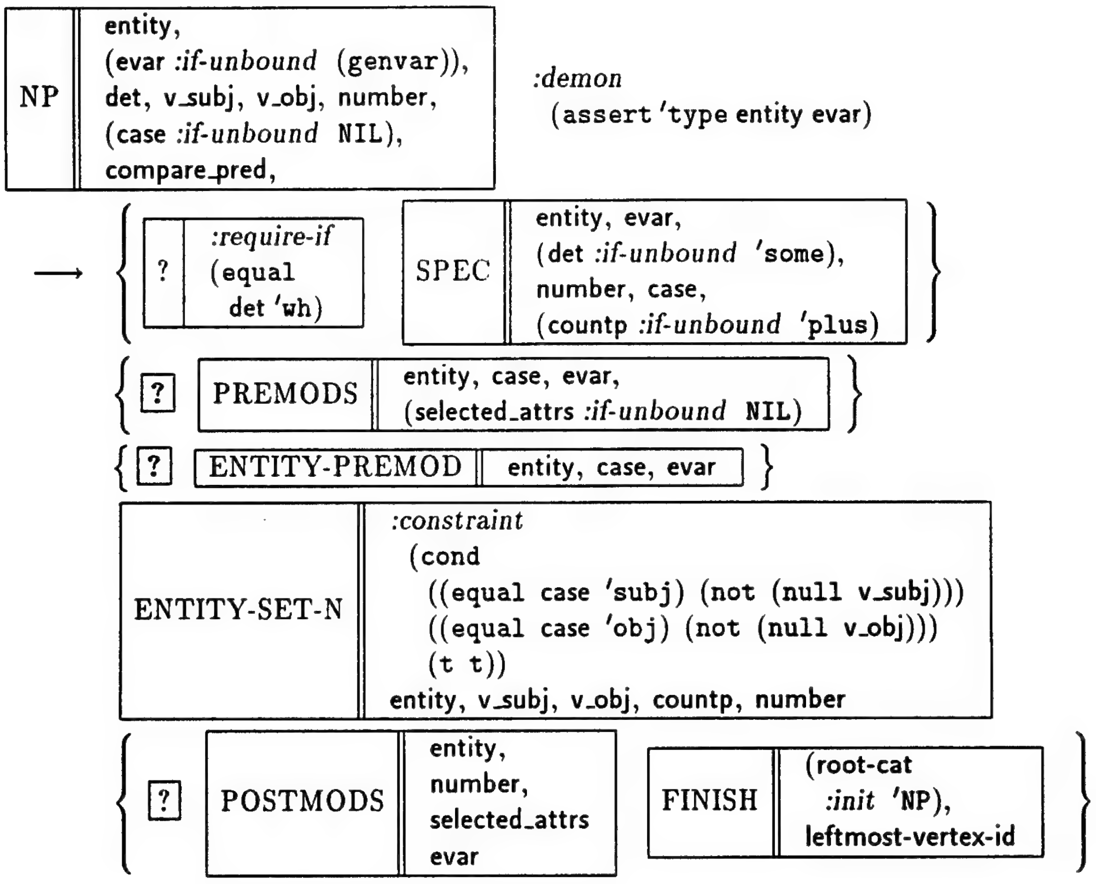
	Event Type		
Decoration Type	N	E	I
:abort-if	√	√	√
:on-exit			√
:demon	√	√	√

The VP rule in Figure 7.4 shows the use of *:abort-if* decorations. This rule enables the user to express entity specialization queries such as "Which theses are Ph.D. theses?" The *:abort* decoration states that the VP is valid only when there exists at least one specialized entity of the subject entity. The *:on-exit* decoration is useful for constructing the binding of a feature variable from other feature variable bindings before returning it to parent edges. The *:demon* decoration is useful for expressing arbitrary actions every time when the state of an activated grammar rule changes. Figures 7.2 and 7.4 show the use of this decoration for passing the partial query information to the rule-based inference system.

7.4.4 Preterminal Category Decoration

While the *:abort-if* decoration provides the power of pruning rule applications, its granularity of pruning is coarse, limited to the whole category. Selective pruning of lexicon entries based on arbitrary filtering constraint is desirable. The preterminal category decoration *:constraint* enables to express conditions for such instance-level pruning:

:constraint $\mathcal{E}$: The expression $\mathcal{E}$ become the filtering constraint. Both *:init* and *:if-unbound* decorations are used to evaluate $\mathcal{E}$.



```
NP || entity,
      (evar :if-unbound (genvar)),
      det, v_subj, v_obj, number,
      (case :if-unbound NIL),
      compare_pred,
                                        :demon
                                          (assert 'type entity evar)

  →  { ? | :require-if      SPEC || entity, evar,
           (equal                   (det :if-unbound 'some),
             det 'wh)               number, case,
                                    (countp :if-unbound 'plus) }

     { ? PREMODS || entity, case, evar,
                    (selected_attrs :if-unbound NIL) }

     { ? ENTITY-PREMOD || entity, case, evar }

     ENTITY-SET-N || :constraint
                       (cond
                         ((equal case 'subj) (not (null v_subj)))
                         ((equal case 'obj) (not (null v_obj)))
                         (t t))
                     entity, v_subj, v_obj, countp, number

     { ? POSTMODS || entity,          FINISH || (root-cat
                     number,                     :init 'NP),
                     selected_attrs              leftmost-vertex-id }
                     evar
```

Figure 7.5: A Grammar Rule on NP

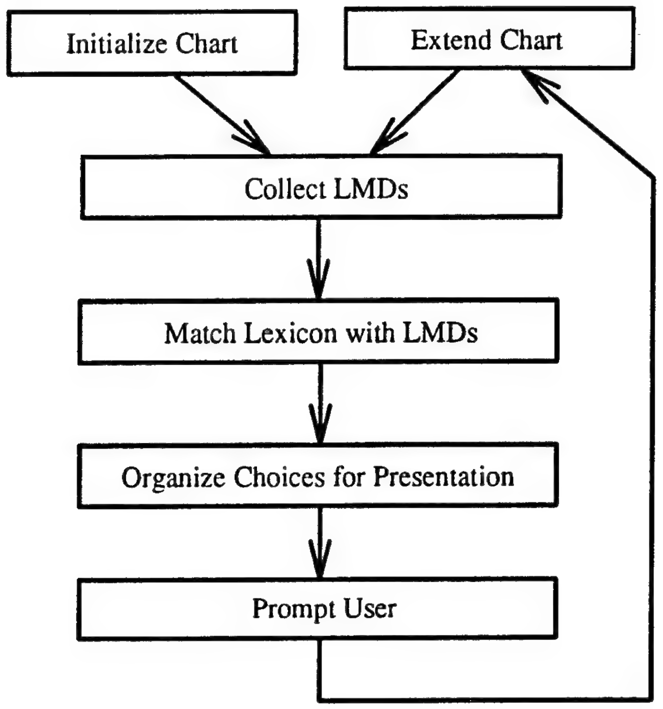


Figure 7.6: Control Flow of Choice Generation Process

The NP rule of Figure 7.5 illustrates the utility of this decoration in pruning dead-end choices. The *:constraint* predicate attached to the preterminal ENTITY-SET-N states that only nouns whose v_subj (v_obj) is not null are presented as menu choices for constructing a subject (an object) NP.

7.5 Control Flow of Choice Generation

The left-to-right incremental query construction in Kaleidoscope fixes the way that active and inactive edges are created on the chart. Figure 7.6 shows the overall control flow of choice generation.

1. *Initialize Chart:* When the user starts a query, the system initializes the chart. This chart initialization involves top-down activation of grammar rules starting

```
RecursiveRuleApply ($cat\ binding\ vertex\ parentEdge$)
begin
if $cat$ is a preterminal symbol
  then CreateActivePreterminalEdge ($cat\ binding\ vertex\ parentEdge$)
  ;; The choice set is computed from the collection of active preterminal edges at vertex.
else begin
    $R \leftarrow$ MatchRules ($cat\ binding$)
    for each $rule \in R$
        do begin
        $edge \leftarrow$ CreateNewActiveEdge ($rule\ binding\ vertex\ parentEdge$)
        $rhs \leftarrow$ GetRuleRHS ($rule$)
        $S \leftarrow$ GetSequenceSet ($rhs\ edge$)
        ;; If rhs begins with one or more optional sequences, the cardinality of $S > 1$.
        for each $seq \in S$
            do begin
            ($cat_{Child}$, $binding_{Child}$) $\leftarrow$ GetFirstTerm ($seq\ edge$)
            RecursiveRuleApply ($cat_{Child}\ binding_{Child}\ vertex\ edge$)
            end do
        end do
end else
end.
```

Figure 7.7: Recursive Top-Down Procedure

```
ExtendChart (inactive vertex)
begin
if EdgeSymbol (inactive) is the start symbol
  then declare inactive is complete at vertex.
  ;; This enables the choice "RUN QUERY."
P ← GetParentEdges (inactive)
for each parent ∈ P
    do begin
    S ← GetRemainingSequenceSet (parent inactive)
    for each seq ∈ S
        if seq is null
          then $inactive_{parent}$ ← InactivateActiveEdge (parent inactive)
                ExtendChart ($inactive_{parent}$ vertex)
        else begin
             $extended_{parent}$ ← ExtendActiveEdge (parent inactive)
             ($cat_{Next}$, $binding_{Next}$) ← GetFirstTerm (seq parent)
             RecursiveRuleApply ($cat_{Next}$ $binding_{next}$ vertex parent)
        end else
    end begin
end.
```

Figure 7.8: Recursive Bottom-Up Procedure

from the start symbol S. For example, given a top-level rule S $\rightarrow$ NP VPS, the interpreter creates new active edges for this rule and the rules on NP, the first RHS symbol. This process applies to each of the rules on NP and continues recursively until preterminal categories are met. The interpreter assumes no left recursion in grammar. (Any grammar with left recursion can be transformed into an equivalent grammar without left recursion.) The procedure RecursiveRuleApply presented in Figure 7.7 captures the process of this top-down recursive rule activation.

2. *Collect LMDs:* The menu choice set is computed from active preterminal edges on the current vertex. Once chart initialization is completed, the choice generator collects a list of lexicon match descriptors (LMDs), one from each active preterminal edge on the current vertex. An LMD defines three slots:

   (a) A category symbol.

   (b) A feature binding constraint.

   (c) An optional constraint predicate derived from *:constraint* decoration.

3. *Match Lexicon:* For each LMD, the choice generator collects the lexicon entries that unify with the given feature binding constraint. If the LMD contains a constraint predicate, the choice generator further filters matched entries with this predicate.

4. *Organize Choices for Presentation:* The collected lexicon entries are regrouped by their menu windows. Each choice string may have multiple preterminal active edges as its source. A choice maintains this source information. Hierarchically organized lexicon entries need special treatment at this step to avoid presentation of choices with different display strings but with identical feature bindings after unification. Consider the LMD whose category specification is ENTITY-SET-N and that includes {entity = *Book*} as a part of its feature binding constraint. The following lists matched choices with the resultant binding of entity:

"references"	{entity = *Book*}
"books"	{entity = *Book*}
"(authored) books"	{entity = *Authored_Book*}

It is unnecessary to present the seemingly general choice "references" to the user when the entity set that it refers to is identical to the entity set that the choice "books" does.

5. *Prompt User:* Collected choices are presented on the menu awaiting the user's selection.

6. *Extend Chart:* The user's choice selection adds a new vertex to the chart. Inactive edges are created for the source preterminal edges of the selected choice. These inactive edges in turn inactivate or extend other active edges in the chart. The procedure ExtendChart in Figure 7.8 captures this bottom-up process. The top-down procedure RecursiveRuleApply is called in this process to create new set of active preterminal edges. When this step is complete, proceed to the step 2.

7.5.1 Dynamics of Chart Manipulation

The major event for the interpreter to monitor is creation of inactive edges immediately on the right of active edges. When a new active edge is instantiated on the chart for a grammar rule, it keeps the information about which active edges are awaiting its inactivated version. When the inactivated version of this active edge is created later, the procedure ExtendChart knows which active edges to consider for extension.

To illustrate the dynamics of extending the chart, let a_k be an active edge awaiting the instantiation of an inactive edge labeled with s, and ${i^s}_1$ be such an inactive edge immediately on the right of a_k. Instead of modifying the active edge a_k to subsume the inactive edge ${i^s}_1$, another active edge a_{k+1}, if the active edge has additional constituents to be subsumed, or an inactive edge i_{a_k} otherwise, is created. In either case, the new edge extends the active edge a_k to subsume the inactive edge ${i^s}_1$. Both active and inactive edges may be created if an optional sequence is the only remaining sequence. The active edge a_k is reused when ${i^s}_2$, another inactive edge labeled with s, is created later immediately on the right of a_k. This makes it possible for multiple hypotheses to be pending at a time.

An edge maintains a binding list of feature variables as well as a reference to the grammar rule. When a child rule is fired by an active edge representing the parent rule, unification is applied to the *:INIT* bindings of the child rule's LHS feature variables and the corresponding bindings inherited from the parent edge. If this succeeds, a new active edge is created for the child rule with its feature variables initialized to the result of unification. The active edge is not created if the unification fails or if the *:abort-if* condition of the child rule is true. Two decorations *:abort-if* and *:demon* are evaluated later on whenever a new version of the active edge is created to subsume an inactive edge immediately on its right. The *:on-exit* decoration is evaluated only when an inactive edge is created. When an inactive edge is created, its feature variable binding list is passed up to a new version of the parent edge. In this process, the binding of the child edge overrules the existing binding of the parent edge.

7.6 Grammar Transformation

Table 7.3 shows the verb forms uncommon in linguistics. A verb of ppby form results from concatenating a verb of pp form and "by." Similarly, a verb of beppby form results from concatenating a BE auxiliary verb, a verb of pp form, and "by." This concatenation is motivated to reduce the number of steps required to create passive verb phrases. Note that the grammar resulting from concatenation is both syntactically and semantically equivalent to the original grammar.

Definition: Two grammars are equivalent if they generate the same set of strings, and the sets of derivations built on each string have the same cardinality and the same set of possible semantic interpretations.

Definition: Two preterminals in a grammar are adjacent if there exist derivations placing them next to each other.

Let A, B be adjacent preterminals in a rule on X, say,

Model Concept	Preterminal Category	Exemplary Choices
Attribute	WHOSE-ATTR-N-BE	whose name is
Subordinate Entity	WHOSE-POSSED-SET-N	whose books

Table 7.4: Alternative Preterminal Categories: Examples

$$X \rightarrow \ldots A, B \ldots$$

Let A_1 and A_2 be new preterminals dividing the lexicon entries in the category A into two disjoint subsets, and A_1B be a new preterminal containing the result of concatenating $a \in A_1$ and $b \in B$ with all possible permitted combinations. Then the above rule is split into

$$X \rightarrow \ldots A_1B \ldots$$

$$X \rightarrow \ldots A_2, B \ldots$$

If A_2 is null, we say that the concatenation is total. Otherwise, it is called partial. This binary concatenation is applicable to an arbitrary number of symbols in sequence.

Table 7.4 shows two additional examples of concatenation useful for reducing the number of steps. In the next chapter, we formulate a condition when this type of concatenation is desirable using a simple cost model of user query production.

7.7 Summary

This chapter has introduced a flexible grammar formalism for choice generation in Kaleidoscope. It extends context-free grammar in two ways: (1) augmentation of symbols with feature attributes and (2) attachment of procedural decoration to grammar rules. The first captures the context of constituent categories, while the latter provides the power of expressing arbitrary constraints and actions in grammar. The choice generator uses the chart as a run-time data structure for representing the partial query state. The parallel and monotonic nature of the chart-based grammar interpretation makes the chart suitable for keeping track of feature bindings. Kaleidoscope's incremental choice generation fixes the way that active and inactive edges are created on the chart. The detailed grammar interpretation process has been described in this chapter. Finally, we have discussed possible ways of transforming a grammar while maintaining the expressive power of the original grammar.

Chapter 8

Quantitative Dimension of Interface Design

The direct public cost of schooling a child for thirteen years, from kindergarten through twelfth grade is over \$20,000 today (and for the class of 2000, it may be closer to \$30,000). A conservatively high estimate of the cost of supplying each of these children with a personal computer with enough power for it to serve the kinds of educational ends described in this book, and of upgrading, repairing, and replacing it when necessary would be about \$1,000 per student, distributed over thirteen years in school. Thus "computer costs" for the class of 2,000 would represent only about 5 percent of the total public expenditure on education, ...

– Seymour Papert, *MINDSTORMS* (1980)

A choice in user interface design potentially affects the performance of end users. Design of a new grammar-driven menu interface is not an exception. A framework for systematic selection of design choices is desired for the grammar-driven menu interface. The normative design principle, which emphasizes the designer's articulation and evaluation of alternative designs [62], is illuminating. To provide a quantitative basis of design evaluation, we present a simple cost model of user query production when using the grammar-driven menu interface.

8.1 Motivation

The grammar design in Kaleidoscope decides two aspects of the interface:

1. A range of syntactic alternatives for a given functionality.

2. The granularity of query constituents presented as choices.

Alternatives exist for each of these aspects, and a design choice potentially affects the performance of end users. We illustrate this point for the entity set restriction via attributes and subordinate entity sets.

Syntactic Coverage An entity set may be restricted via either prenoun or postnoun modifiers:

- "DATABASE" in "DATABASE books" (a prenoun modifier).

- The clause following "books" in "books whose keyword is DATABASE" (a postnoun modifier).

The postnoun modifier alone suffices to provide the functionality of entity set restriction. However, the prenoun modifier alone does not because some attributes such as *Book*'s *id* are unqualified for the prenoun modifier. The prenoun modifier is not an alternative but an extraneous syntactic feature.

The NLMenu grammar for database access favors minimizing the space of user choices, thus leaving out the prenoun modifier feature. However, as the query Q1 illustrates, this feature enables the user to express concise queries. The negative side of including an extraneous feature such as the prenoun modifier in the syntax is that choices based on such a feature incur a pure overhead to the user who is not interested in selecting them.

Granularity of Constituent Selection The NLMenu grammar for database access in general takes a large constituent unit as a single menu choice. This has two advantages. First, it reduces the number of steps necessary to express a query. Second, a choice is more likely to be rich in its semantic content. For example, NLMenu presents "whose book keyword is" as a single choice for leading a postnoun modifier clause. In contrast, INGLISH and NLParse/NLGen adhere to word-by-word presentation. In these systems, "whose," "keyword," and "is" are presented as separate choices in a sequence.

Norman's Slogan In making design choices, one of Norman's slogans in the user interface design [48] is instructive:

> *There are no simple answers, only tradeoffs.* A central theme of our work is that, in design, there are no correct answers, only tradeoffs. Each application of a design principle has its strength and weakness; each principle must be interpreted in a context. One of our goal is to make the tradeoffs explicit.

To follow this slogan, we need to develop a cost model for design evaluation. In the next section, we devise a simple cost model of user query production based on previous human factors experiments on menu systems.

8.2 Cost Model of User Query Production

The cost of user query production is the sum of the cost incurred by the menu states encountered by the user:

$$C = \sum_{i=0}^{l} C_i = \sum_{i=0}^{l} (M_i + F + S_i)$$

where l: the number of steps required to create a query,
M_i: machine-dependent cost at the i-th step,
F: user fixed cost at each step,
S_i: user search cost.

All costs are measured in elapsed time.

8.2.1 Non-search Cost

The machine-dependent cost may vary by the state of the partial query, the number of grammar rules applicable for extending the partial query, and the number of choices to display. This cost is scalable by the MIPS of the machine. While we can actually measure M_i over the machine, we assume a nominal value M for M_i for the simplicity of cost computation. For the XEROX 1186 lisp machine which runs at 0.75 MIPS, M is roughly 7 ([3 - 11]) seconds for the bibliographic domain. For the 15 MIPS machine, this scales down to 0.35 ([0.15 - 0.55]) second. For an ideal machine with the infinite execution speed, $M = 0$.

The fixed cost F accounts for the user's nonvisual search cost incurred by various perceptual and motor actions at each step, such as moving the mouse to the target choice and clicking the mouse. F was measured approximately 1.2 seconds by previous human factors experiments [11, 10].

8.2.2 Search Cost

A model of search cost should capture:

- As users get familiar with the system, they locate intended items without the exhaustive search [10].
- The user often backtracks previously selected choices. We suspect that the more restrictive a grammar is, the more often users backtrack choices.

We model the search cost at each step as follows:

$$S_i = \alpha(1+\rho)\{n_{iS} + \beta(n_i - n_{iS})\}$$

where n_i: the total number of choices at the $i-$th step

n_{iS}: the number of choices in the selected menu window at the $i-$th step

α: the unit cost incurred per choice

$0 \leq \beta \leq 1$: the user inexperience factor

$\rho \geq 0$: the loss factor caused by the user's backtracking.

We assume that each choice item incurs the same cost regardless of its string length. The constant α represents this unit cost. In choosing the target item, we also assume that other choices in the selected menu window incur the same cost as the selected item. This gives the minimal cost proportional to n_{i_S}, the number of choices in the selected menu window. The case of maximal cost occurs for the user's exhaustive search. The inexperience factor β parametrizes the cost between two extremes. The closer β is to 1, the higher the search cost is. The parameter ρ is the loss factor that accounts for the relative portion of choices that are consumed by the user's backtracking. We assume that ρ is dependent on a grammar. This leads us to define:

$$\alpha' \equiv \alpha(1+\rho)$$

The value of α

We estimate the unit cost α by combining the results of two human factors experiments on the menu interface.

1. Card assumed a nonsystematic model of choice search for the fixed-size menu and measured the distribution function of search time [10]. The nonsystematic model assumes that the user has no means of keeping track of where the user has searched, and thus the user's choice search time t follows an exponential distribution:

$$F(t) = 1 - \exp^{-mt}$$

$$E(t) = \frac{1}{m}$$

   Here $F(t)$ and $E(t)$ denote the cumulative distribution function and the expectation of the search time, respectively. The constant m is determined by the number of saccades made for each menu selection and the probability of finding a target item during each eye fixation. For the menu of $n = 18$ items, Card measured $m \approx 1.2$.

2. Perlman measured the effect of menu length on the user's search time [51]. The result of his experiment suggests that the search time is a linear function of the number of choices.

By combining the results of these two experiments, we compute α as follows:

$$\alpha = (\frac{1}{m})\,(\frac{1}{n}) = (\frac{1}{1.2})\,(\frac{1}{18}) = 46(\text{msec})$$

In the rest of this chapter, we illustrate the use of the cost model developed in this section.

8.3 An Illustrative Example: Effect of Concatenation

An equivalent grammar results from concatenating choices in adjacent preterminal categories. This section, taking the example of postnoun modifiers, formalizes the effect of concatenation on the cost of user query production, and formulates the condition under which concatenation is desired.

8.3.1 Problem

Let ΔG_1 be a portion of the grammar with the following rules on postnoun modifiers:

$$\begin{array}{lll} \text{POSTMOD} & \rightarrow & \text{WHOSE ATTR-N-BE ATTR-MOD-EXPR} \\ \text{POSTMOD} & \rightarrow & \text{WHOSE POSSED-SET-N VPS}_{pp} \end{array}$$

These rules prescribes the user's construction of postnoun modifiers for restricting entity sets with its attributes and subordinate entity sets, respectively.

Let ΔG_2 be an equivalent grammar of ΔG_1 with the following rules on POSMOD:

$$\begin{array}{lll} \text{POSTMOD} & \rightarrow & \text{WHOSE-ATTR-N-BE ATTR-MOD-EXPR} \\ \text{POSTMOD} & \rightarrow & \text{WHOSE-POSSED-SET-N VPS}_{pp} \end{array}$$

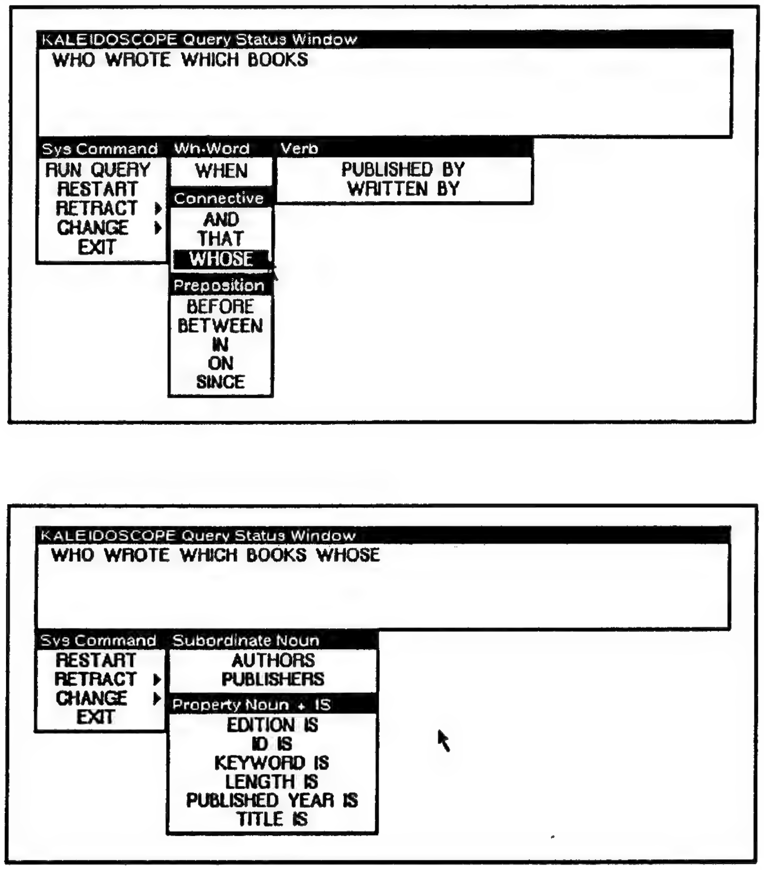


Figure 8.1: Two Consecutive Menu States Based on ΔG_1 (No Concatenation)

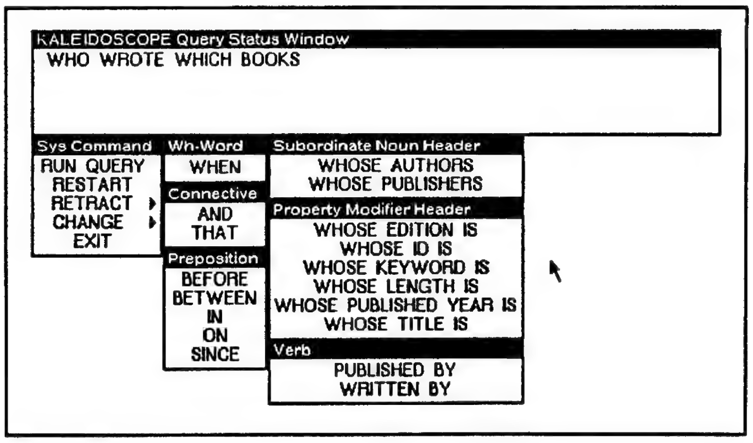

WHO WROTE WHICH BOOKS

Figure 8.2: Menu State Based on ΔG_2 (Concatenated Grammar)

The preterminal category WHOSE-ATTR-N-BE is a category formed by concatenating WHOSE and ATTR-N-BE in ΔG_1. Similarly, the category WHOSE-POSSED-SET-N is formed by concatenating WHOSE and POSSED-SET-N. The following illustrates the content of the lexicon corresponding to two grammars:

Grammar	Category	Example String
ΔG_1	WHOSE	"whose"
	ATTR-N-BE	"title is"
	POSSED-SET-N	"publishers"
ΔG_2	WHOSE-ATTR-N-BE	"whose title is"
	WHOSE-POSSED-SET-N	"whose publishers"

Figure 8.1 shows two consecutive menu states based on the grammar ΔG_1 while Figure 8.2 shows a corresponding state based on the grammar ΔG_2. This example shows that choice concatenation reduces the number of steps necessary to formulate queries matching certain patterns, but also increases the number of choices for other paths. For example, if the user is selecting a preposition, the choices in the windows "Subordinate Noun Header" and "Property Modifier Header" create a pure overhead.

8.3.2 Tradeoff Formulation

Let p_{pmod} be the probability of choosing a postnoun modifier clause after selection of a noun for representing an entity set, n_{pmod} be the total number of the selected noun's subordinate entities and properties, and n_r be the number of choices that are not affected by ΔG_i's ($i = 1, 2$). For the menu state shown in Figures 8.1 and 8.2, $n_{\text{pmod}} = 8$ and $n_r = 15$ (5 system commands, 1 *wh*-word, 2 connectives, 2 verbs, and 5 prepositions).

We consider the case of $\beta = 1$ (exhaustive search). The case of $\beta = 0$ is trivial; the concatenated grammar always wins. Let

$$\Delta C_2 = (M + F) + \alpha'(n_{\text{pmod}} + n_p).$$

Then ΔC_2 is the cost incurred by the menu state of Figure 8.2. The comparable cost incurred by the menu states of Figure 8.1 is given by:

$$\begin{aligned} \Delta C_1 &= (M + F) + \alpha'(n_r + 1) \\ &+ p_{\text{pmod}} \{(M + F) + \alpha' n_{\text{pmod}}\}. \end{aligned}$$

The right-hand side of the first line accounts for the cost incurred by the first menu state, while the second line accounts for the cost by the second menu state. The latter is weighed by the probability of choosing "whose" from the first menu state.

Concatenation is desirable if $\Delta C_1 - \Delta C_2 > 0$:

$$\begin{aligned} \Delta C_1 - \Delta C_2 &= \alpha'\{(n_r + 1) + p_{\text{pmod}} n_{\text{pmod}} - (n_{\text{pmod}} + n_r)\} \\ &+ p_{\text{pmod}} (M + F) \\ &> 0 \end{aligned}$$

or

$$p_{\text{pmod}} > \frac{\alpha'(n_{\text{pmod}} - 1)}{\alpha' n_{\text{pmod}} + (M + F)}$$

Interpretation

Notice that this inequality suggests that for slow machines, where M is large, concatenation is desirable even with a low probability of choosing postnoun modifier clauses.

To illustrate, we substitute the parameters $n_{\mathrm{pmod}} = 8, n_r = 15, \rho = 0, F = 1.2$ and vary the value of M:

M = 0:	$p_{\mathrm{pmod}} > 0.66$
M = 0.7:	$p_{\mathrm{pmod}} > 0.58$
M = 7:	$p_{\mathrm{pmod}} > 0.27$

8.4 Summary

This chapter has presented a cost model of the user's query production and illustrated how this model can guide the systematic choice of design options. The proposed model may be used for building an adaptive interface system. Such an interface suggest the user to switch to an alternative grammar design based on the statistics on the actual interface usage and incurred cost.

Chapter 9

Conclusion

Formulating database queries is a cognitively demanding process. In the absence of proper guidance from the interface, the user is burdened to learn and recall precisely the query language and the underlying database. This dissertation has addressed relieving the user of this cognitive burden as the prime function of a cooperative query interface, and presented Kaleidoscope's approach to the design of such an interface system.

9.1 Kaleidoscope's Interface Approach

Kaleidoscope provides an English-like query language for users to phrase queries with restricted yet common English expressions. A grammar-driven menu system bridges the inevitable mismatch between this language and the user's language. By generating legitimate EnQL constituents step by step as menu choices, this matching device relieves casual database users of learning and recalling the restrictions on EnQL and the conceptual coverage of a specific database. Users formulate queries by recognizing choices coming one after another that match their mental queries. The system uses its knowledge actively to guide users to create unambiguous and meaningful queries.

9.2 Central Theme: Model-Based Approach

Kaleidoscope's interface approach reduces the design of a limited language interface to a tractable engineering problem. As the interface takes the initiative in user-system communication, it is possible to apply the normative design principle. The design goal is explicitly represented and alternative designs are evaluated.

The central theme of this thesis is that a data model plays a critical role in designing a normative query interface system, as a query language essentially conveys the underlying conceptualization of data to the user. The design of grammar, lexicon, and query translator follows a well-defined data model. First, grammar design focuses on unambiguous, meaningful realization of references to model concepts. Then, a set of domain-independent procedures is defined for automatically generating lexicon from the schema and a collection of English terms referring to the schema terms. A data model also serves as the basis of implementing a domain-independent mapping to the underlying storage model in the query translator. Thus the creation of an interface over a specific database involves only defining schema terms and English references to such terms. This way of using the data model in Kaleidoscope differs significantly from the way that the model is used in the conventional NLI design, in which the model assists the linguistic processor to assign possible semantics to arbitrary, unconstrained queries.

9.3 Technical Contributions

Data Model for EnQL The major technical contribution of this dissertation is a semantically rich data model for supporting the generation of legitimate English constituents on the menu. As the grammar defines the mapping between the data model and a query language, the simplicity of this mapping has been a primary concern in defining the conceptual primitives. The model defines entities, relationships, and relationship modifiers as basic concepts corresponding to noun phrases, verb phrases, and adverb phrases, respectively. *ISA* hierarchies organize these concepts by similarity and difference. Kaleidoscope uses this hierarchical information in various ways.

The model introduces *I-OVERLAP* to represent the relationship among inherently overlapping entity sets. *I-OVERLAP* avoids the need for explicitly representing entity sets with multiple inheritance, thus reducing the total number of entity sets in the schema. Finally, the model supports the definition of two derived concepts: subordinate entities from relationships and derived attributes from relationship modifiers. They are useful for referring to source concepts concisely.

EnQL, based on the rich set of conceptual primitives, provides users with various degrees of freedom in query formulation. To measure the gain in the user's benefit of using EnQL, we have relied on a syntactic measure – the number of tokens required to express a query. When SQL is taken as a reference, EnQL queries are significantly more concise than their SQL translations, often by an order of magnitude.

Cost Model for Evaluation of Design Alternatives To provide a complete normative design framework, this dissertation has presented the quantitative dimension of grammar-driven menu interface design. We have constructed a cost model of user query production from the result of previous human factors experiments, and illustrated the utility of this model for evaluation of alternative grammar designs.

Flexible Grammar Formalism for Choice Generation In developing a grammar formalism for incremental constituent generation, its flexibility in capturing constraints, both linguistic and heuristic, has been a prime concern as well as the generality of grammar. Following other grammar formalisms for natural language processing, Kaleidoscope extends context-free grammar by augmenting category symbols with feature attributes. In addition, it provides a few types of procedural decoration for attachment to augmented grammar rules. Procedures in these decorations are executed when matching events occur during grammar interpretation. This procedural decoration gives a full degree of flexibility in enforcing the integrity of a partial query and interfacing with the nonlinguistic part of the system.

Engineering Solution to Big Menu Problem The explosion of choices is often cited as the major problem in applying grammar-driven menu guidance to a large application domain. We have articulated a set of heuristics useful for alleviating this problem:

- Restrict the initial set of choices to a small set of words loaded with multiple semantics, such as *wh*-words.
- Activate domain-specific semantics to prune irrelevant choices in the choice generation process.
- Use abstraction hierarchies (*ISA* and *Part-of*) to cluster related choices under a single choice.
- Break a preterminal category formed by concatenating preterminal categories if applicable. This may increase the number of steps to construct a query. The cost of user query production is useful for formulating a trade-off condition when it is desirable.
- As the last resort, construct a subset lexicon based on a subset view of schema.

9.4 Future Research

This research can be extended in several directions.

Beyond Conjunctive Queries This research restricted EnQL to the conjunctive class of queries. This restriction enabled us to focus on the model-based approach without being involved in the sophisticated linguistic issues such as quantifiers and negated verbs. We expect to extend both the surface and internal query languages to include such features. The capability of the underlying query processing system needs to be extended accordingly.

Human Factors Experiment Controlled measurement of the end user's performance with grammar-driven menu system is a challenging work. For this experiment to be meaningful, the current implementation of Kaleidoscope needs to be ported on fast machines. The state of hardware and software technology is mature enough to provide subsecond update in the menu.

While previous human subject experiments involved two styles of languages [56, 73, 33], we expect to compare four cases resulting from combining two types of interface and two styles of language:

Menu-Guided EnQL	Menu-Guided SQL
Conventional NLI	Conventional SQL

Through these experiments, we expect to measure semantic gains such as the user's conceptual freedom in expressing a query, and improve the cost model of user query production.

Towards A Multilingual Interface English is the model language considered throughout this dissertation. We expect to test the universality of Kaleidoscope's interface approach and its EnQL data model against other languages. Especially, we are interested in languages whose origin is different from English, such as Korean, and Japanese.

Application to Other Domains The scope of this work has been guiding the user's query formulation. While this dissertation has taken examples from bibliographic and academic database applications, its approach is not bound to a particular application domain. In the future, we expect to apply Kaleidoscope's approach to other large-scale applications such as integrated design and manufacturing databases [27, 50].

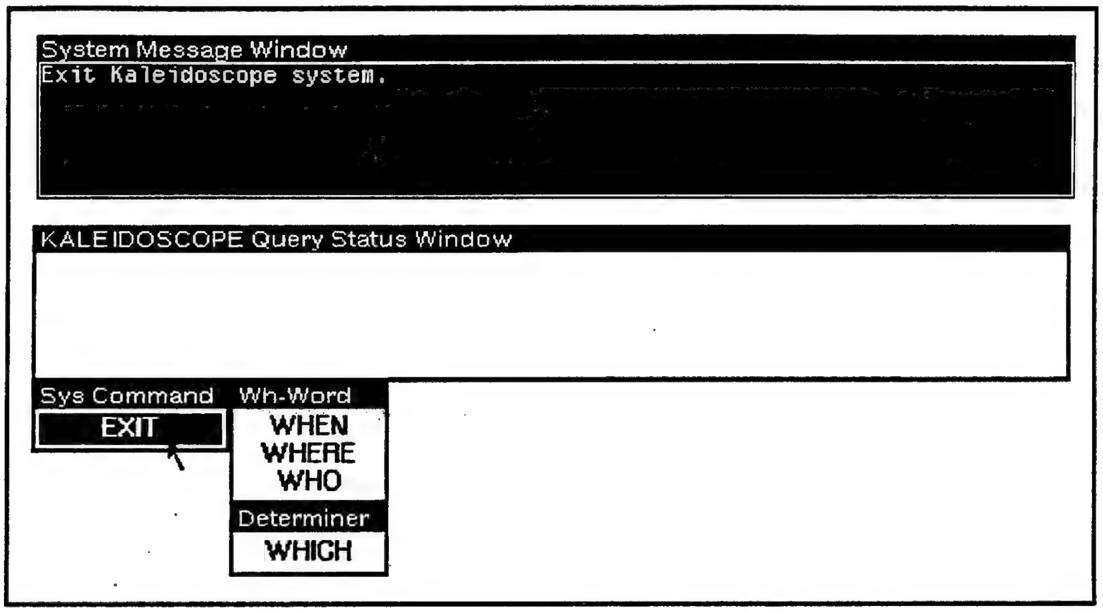
Grammar-driven menu guidance is applicable to other problem domains. Acquisition of production rules is one of such domains. Kaleidoscope's model-based approach is also applicable to this problem domain. The generation of domain-specific vocabularies from the concepts in the knowledge base can be automated.

Appendix A

Complete Sequence of

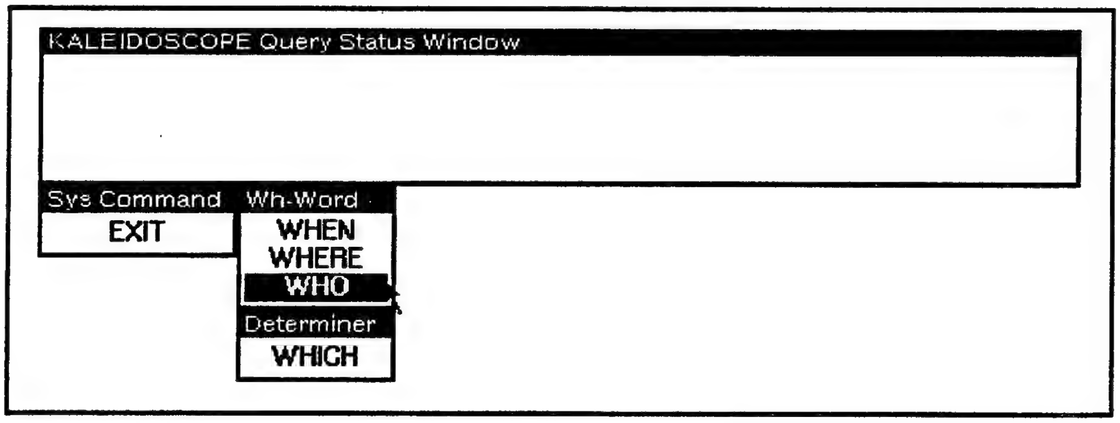
Kaleidoscope States

Initial Screen State

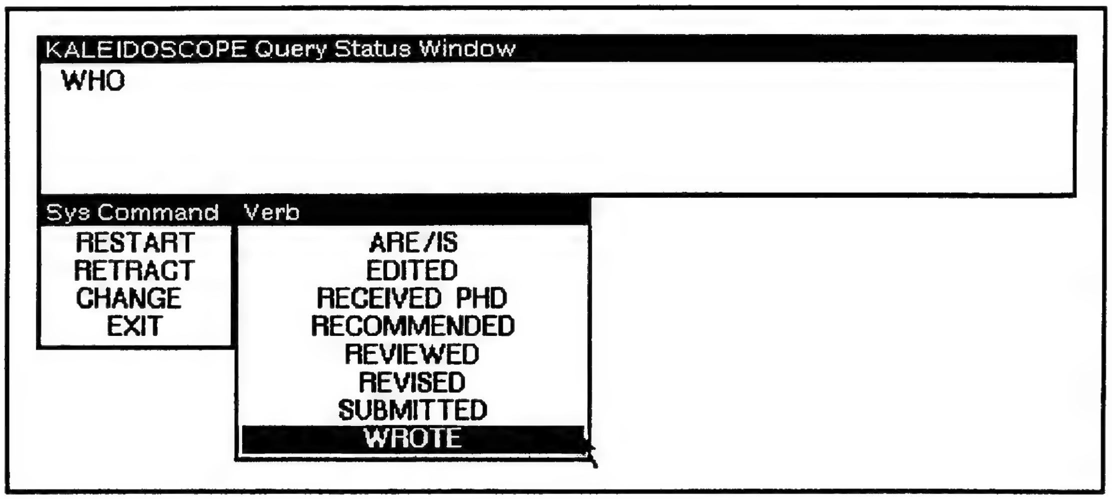


```
Exit Kaleidoscope system.
```

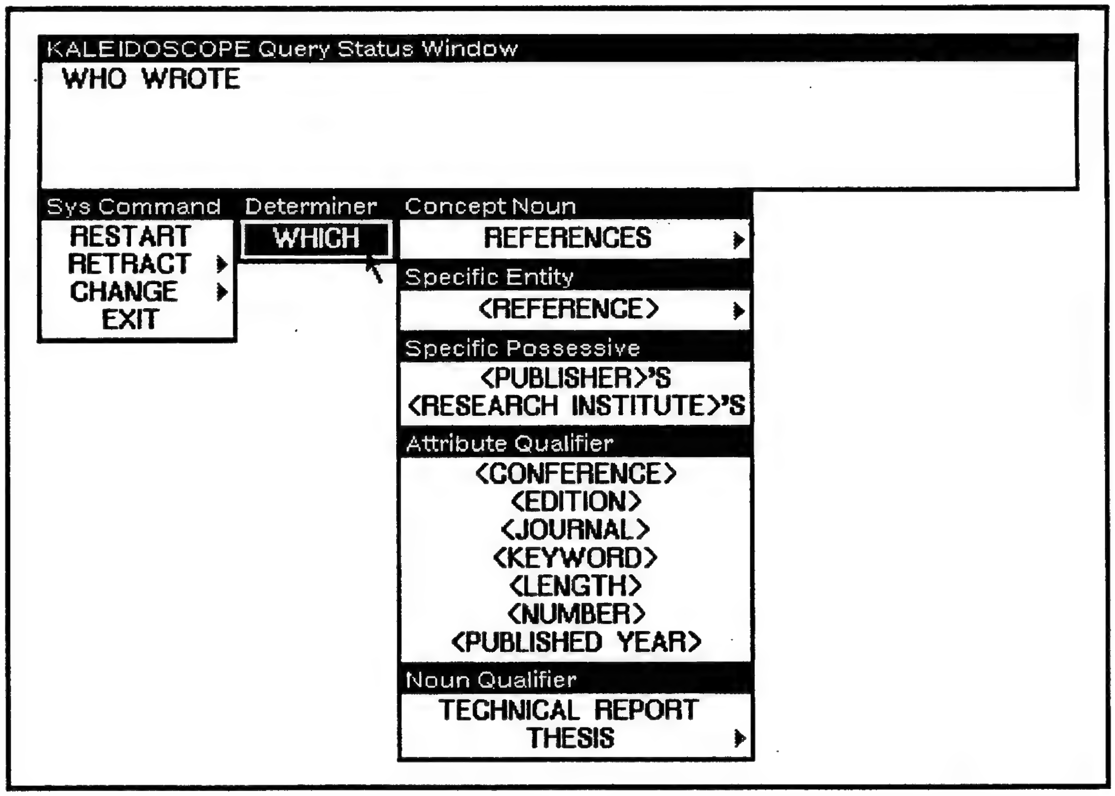
N = 1



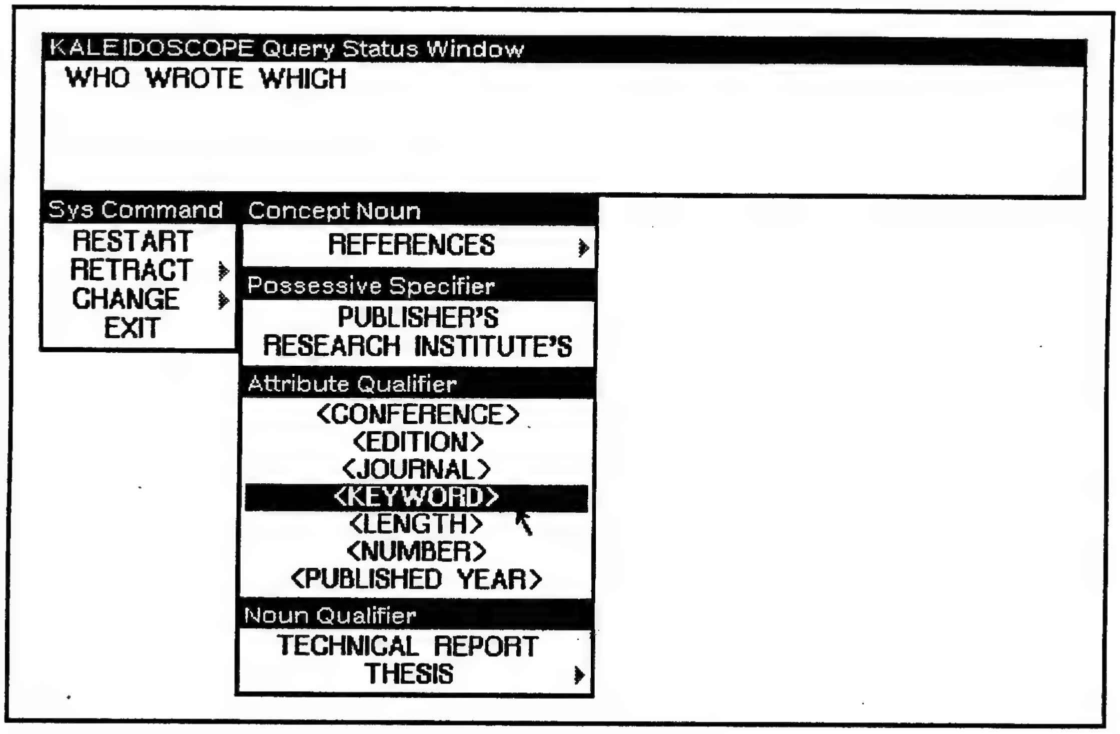
N = 2


KALEIDOSCOPE Query Status Window
WHO
Sys Command
RESTART
RETRACT
CHANGE
EXIT
Verb
ARE/IS
EDITED
RECEIVED PHD
RECOMMENDED
REVIEWED
REVISED
SUBMITTED
WROTE

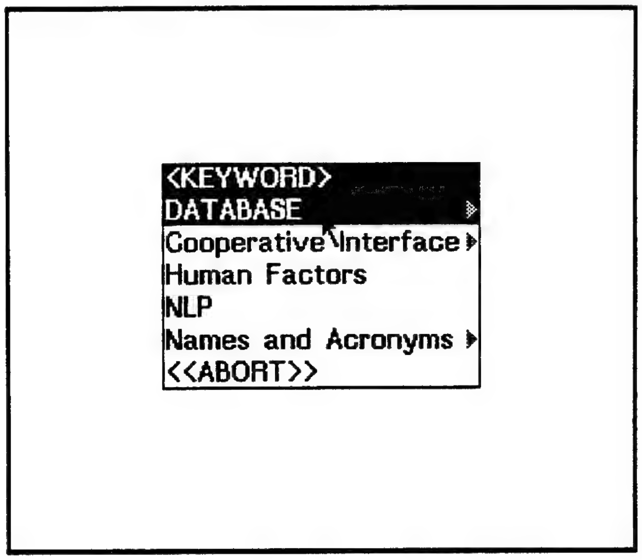
N = 3


KALEIDOSCOPE Query Status Window
WHO WROTE
Sys Command
RESTART
RETRACT
CHANGE
EXIT
Determiner
WHICH
Concept Noun
REFERENCES
Specific Entity
<REFERENCE>
Specific Possessive
<PUBLISHER>'S
<RESEARCH INSTITUTE>'S
Attribute Qualifier
<CONFERENCE>
<EDITION>
<JOURNAL>
<KEYWORD>
<LENGTH>
<NUMBER>
<PUBLISHED YEAR>
Noun Qualifier
TECHNICAL REPORT
THESIS

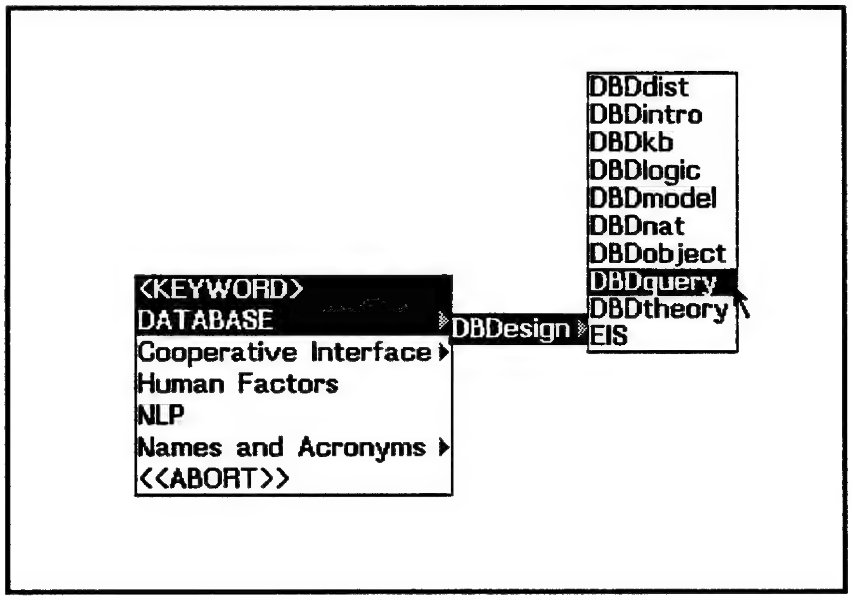
N = 4


KALEIDOSCOPE Query Status Window
WHO WROTE WHICH
Sys Command
RESTART
RETRACT
CHANGE
EXIT
Concept Noun
REFERENCES
Possessive Specifier
PUBLISHER'S
RESEARCH INSTITUTE'S
Attribute Qualifier
<CONFERENCE>
<EDITION>
<JOURNAL>
<KEYWORD>
<LENGTH>
<NUMBER>
<PUBLISHED YEAR>
Noun Qualifier
TECHNICAL REPORT
THESIS

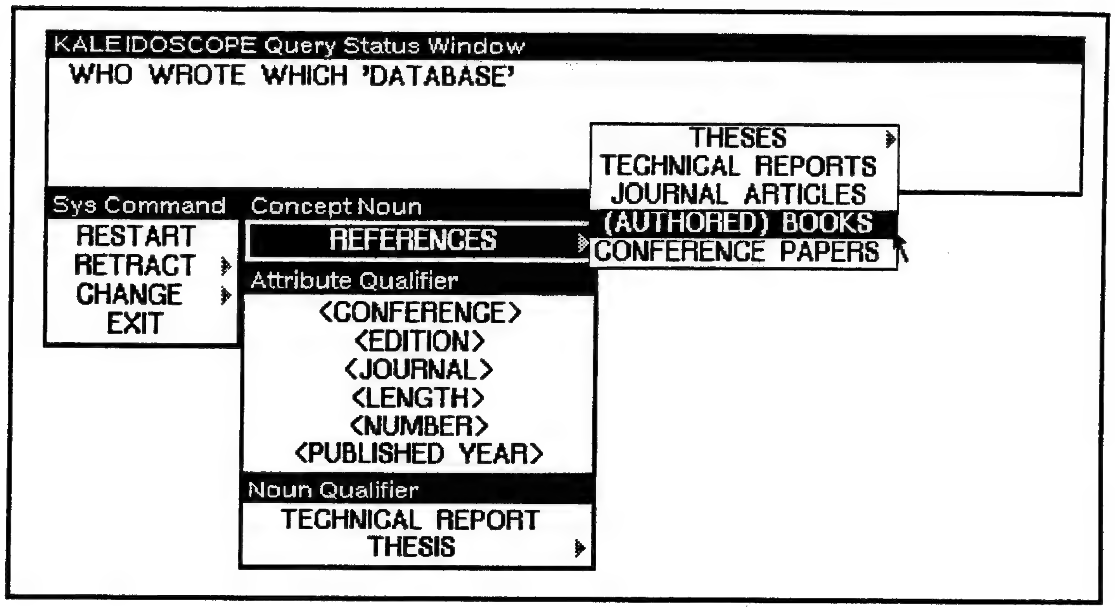
N = 4: Pop-Up Menu for Keyword Value Selection



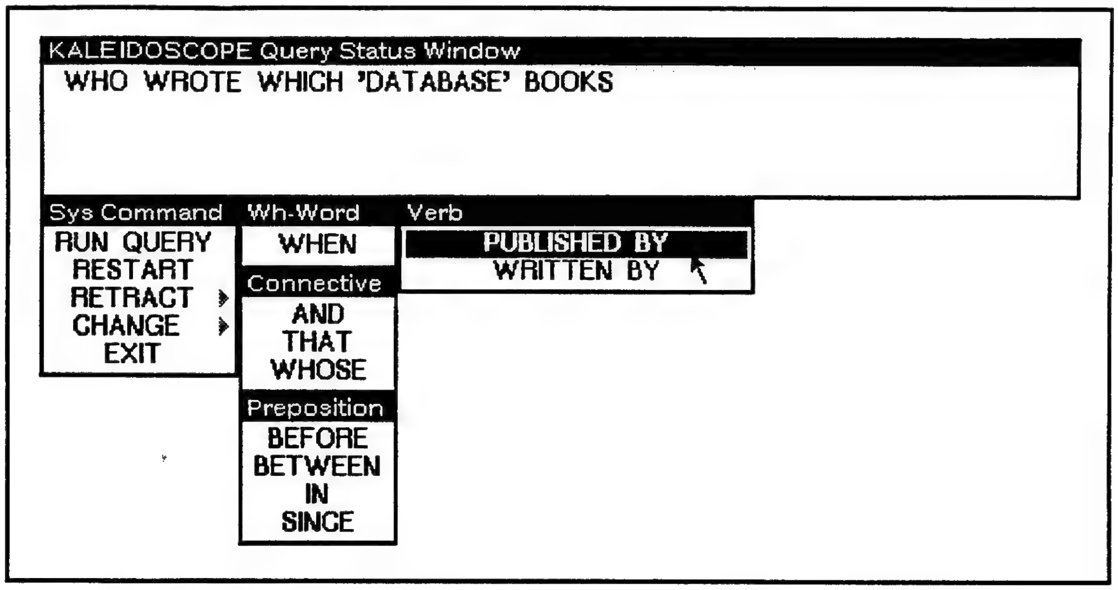
N = 4: Extended Pop-Up Menu State



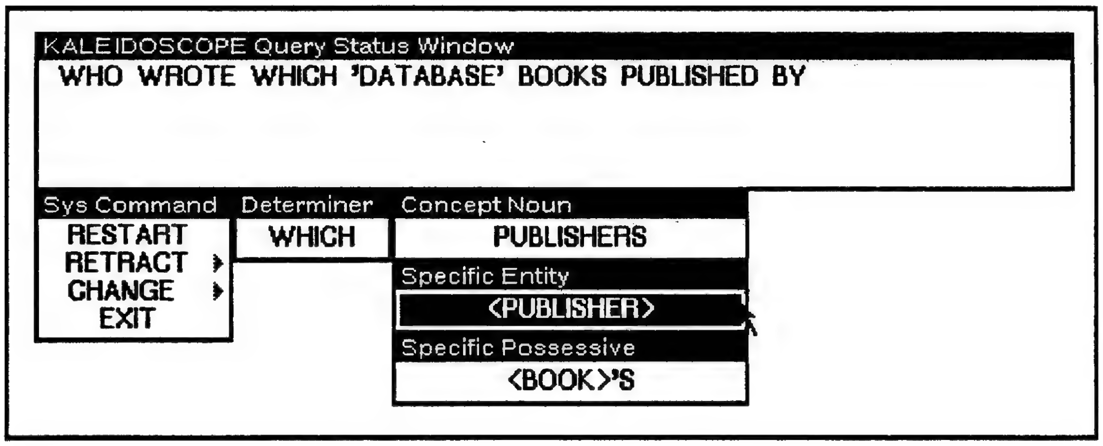
N = 5


KALEIDOSCOPE Query Status Window
WHO WROTE WHICH 'DATABASE'
THESES
TECHNICAL REPORTS
JOURNAL ARTICLES
(AUTHORED) BOOKS
CONFERENCE PAPERS
Sys Command
RESTART
RETRACT
CHANGE
EXIT
Concept Noun
REFERENCES
Attribute Qualifier
<CONFERENCE>
<EDITION>
<JOURNAL>
<LENGTH>
<NUMBER>
<PUBLISHED YEAR>
Noun Qualifier
TECHNICAL REPORT
THESIS

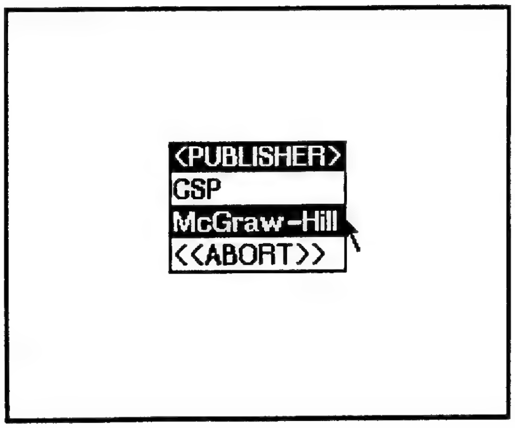
N = 6


KALEIDOSCOPE Query Status Window
WHO WROTE WHICH 'DATABASE' BOOKS
Sys Command
RUN QUERY
RESTART
RETRACT
CHANGE
EXIT
Wh-Word
WHEN
Connective
AND
THAT
WHOSE
Preposition
BEFORE
BETWEEN
IN
SINCE
Verb
PUBLISHED BY
WRITTEN BY

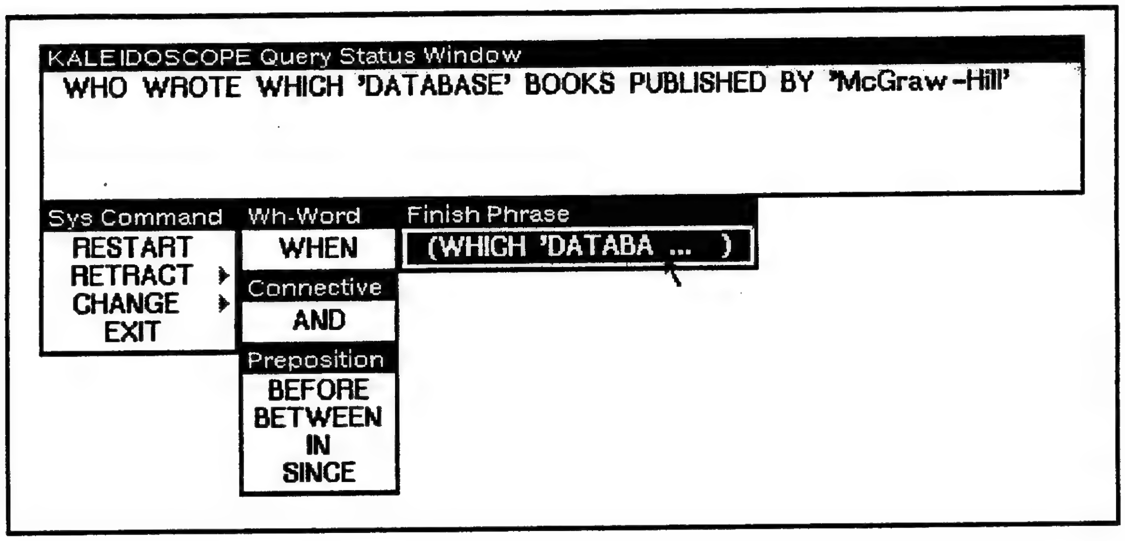
N = 7


KALEIDOSCOPE Query Status Window
WHO WROTE WHICH 'DATABASE' BOOKS PUBLISHED BY
Sys Command
RESTART
RETRACT
CHANGE
EXIT
Determiner
WHICH
Concept Noun
PUBLISHERS
Specific Entity
<PUBLISHER>
Specific Possessive
<BOOK>'S

N = 7: Pop-Up Menu for Publisher value Selection

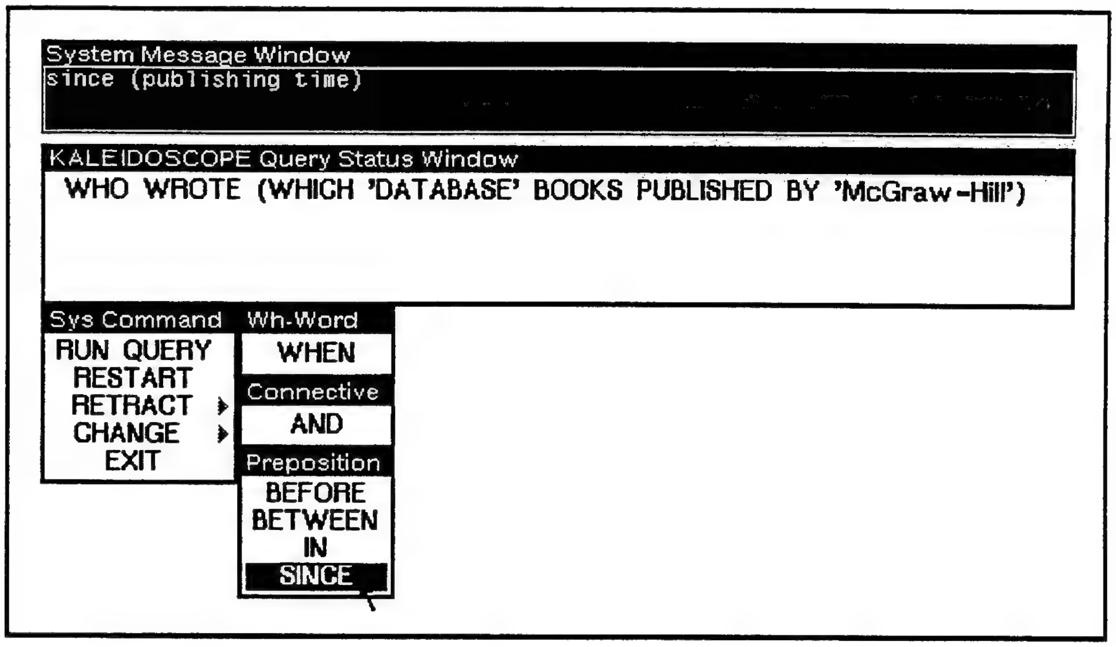


N = 8



WHO WROTE WHICH 'DATABASE' BOOKS PUBLISHED BY 'McGraw-Hill'

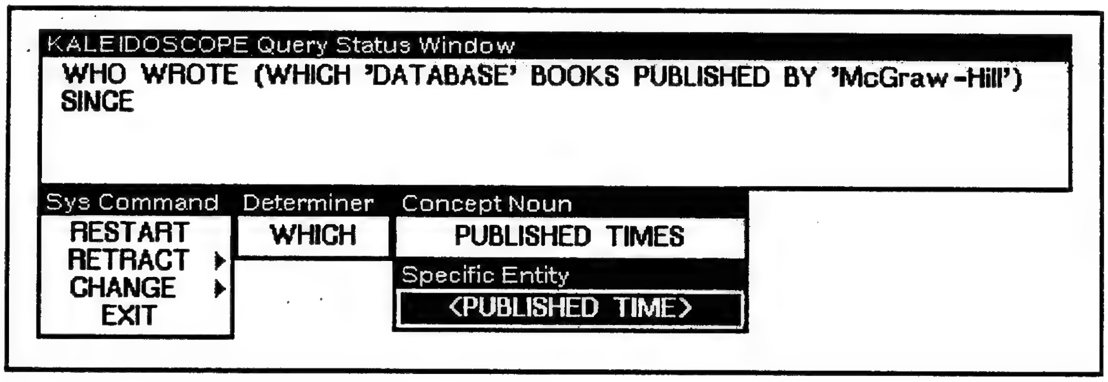
N = 9



```
since (publishing time)
```

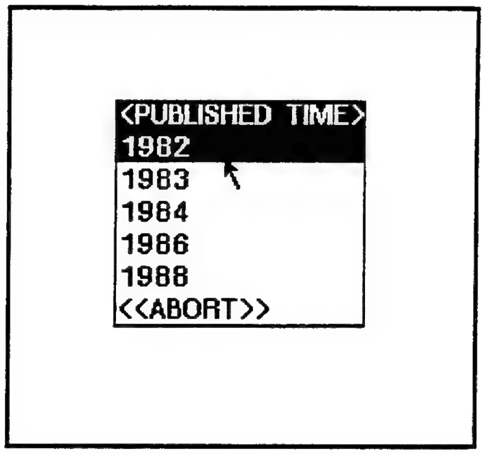
WHO WROTE (WHICH 'DATABASE' BOOKS PUBLISHED BY 'McGraw-Hill')

N = 10

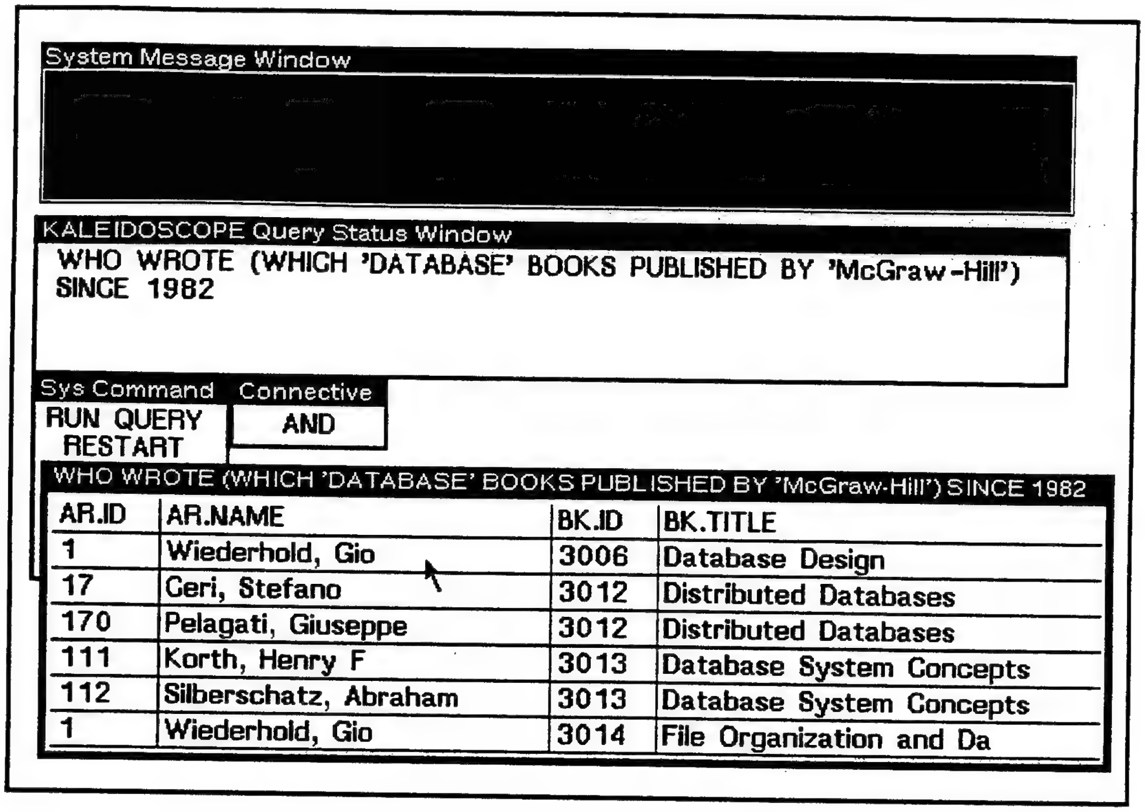


WHO WROTE (WHICH 'DATABASE' BOOKS PUBLISHED BY 'McGraw-Hill')
SINCE

N = 10: Pop-Up Menu for Published Year Selection



Final Screen State



System Message Window

KALEIDOSCOPE Query Status Window

WHO WROTE (WHICH 'DATABASE' BOOKS PUBLISHED BY 'McGraw-Hill') SINCE 1982

Sys Command: RUN QUERY, RESTART

Connective: AND

WHO WROTE (WHICH 'DATABASE' BOOKS PUBLISHED BY 'McGraw-Hill') SINCE 1982

AR.ID	AR.NAME	BK.ID	BK.TITLE
1	Wiederhold, Gio	3006	Database Design
17	Ceri, Stefano	3012	Distributed Databases
170	Pelagati, Giuseppe	3012	Distributed Databases
111	Korth, Henry F	3013	Database System Concepts
112	Silberschatz, Abraham	3013	Database System Concepts
1	Wiederhold, Gio	3014	File Organization and Da

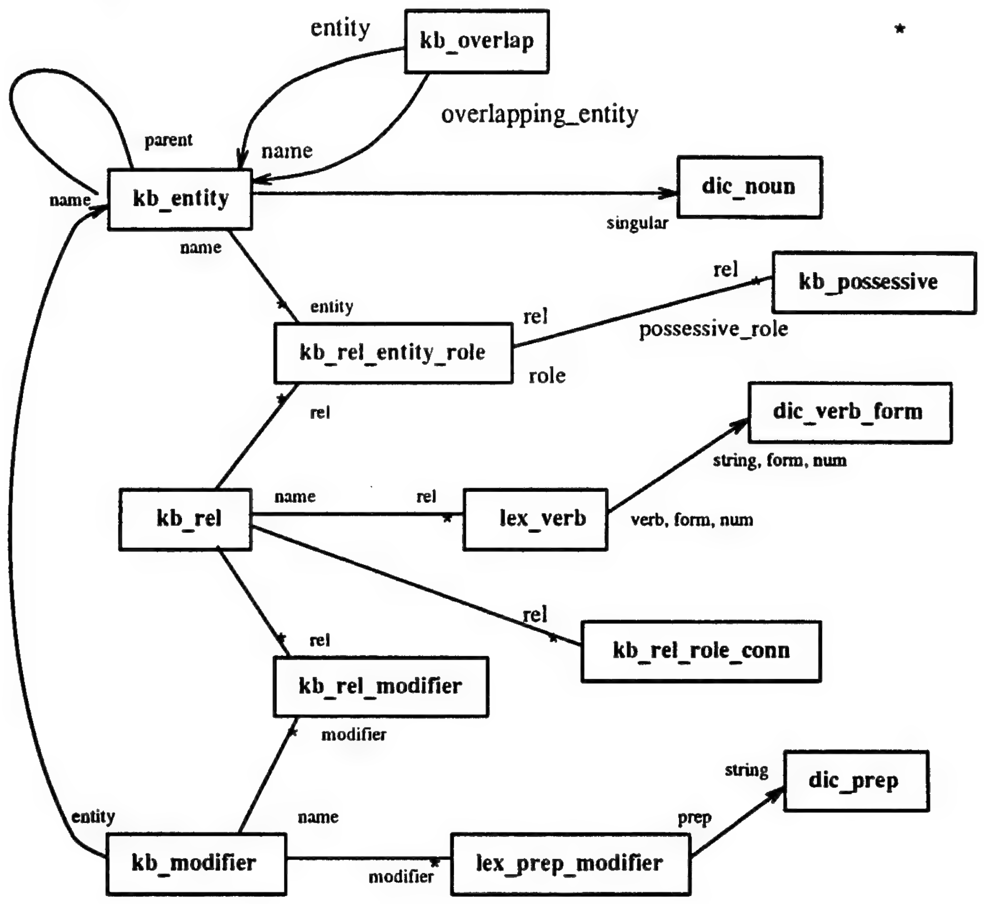
Appendix B

Tabular Representation of Knowledge

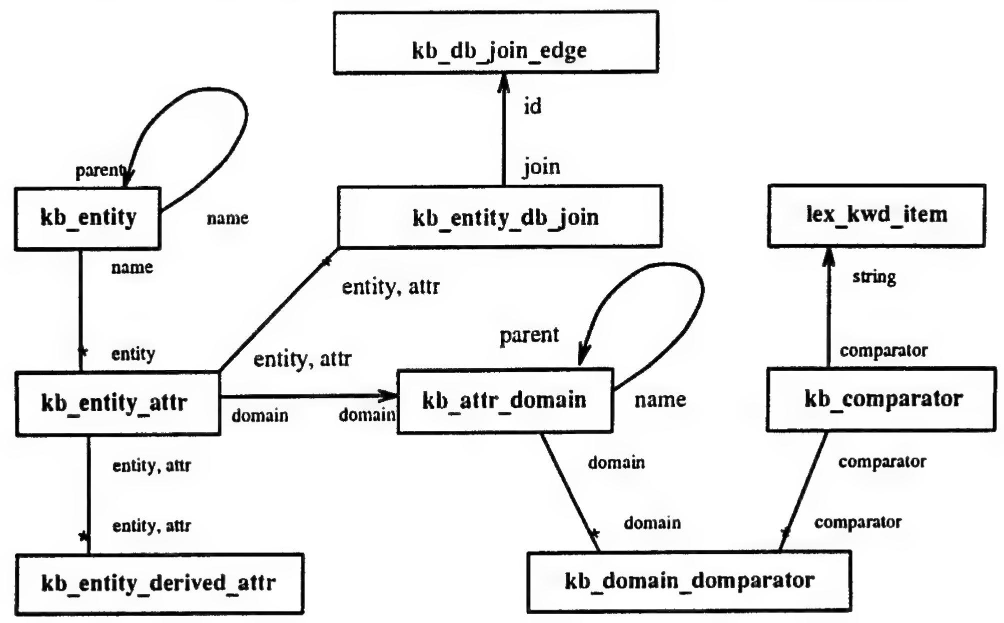
This part of appendix shows the definition of tables for representing Kaleidoscope's knowledge structure. Connection types in the structural data model [75] are used to denote the relationships among these tables in the graphical schema representation.

B.1 Graphical Schema by Related Table Groups

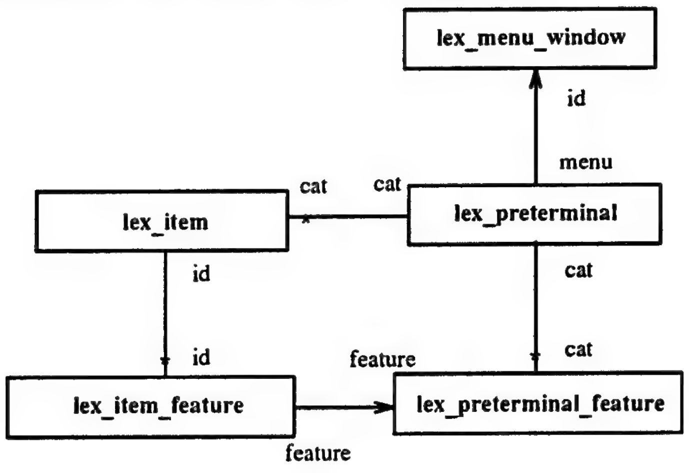
B.1.1 G1: Entities, Relationships, and Relationship Modifiers



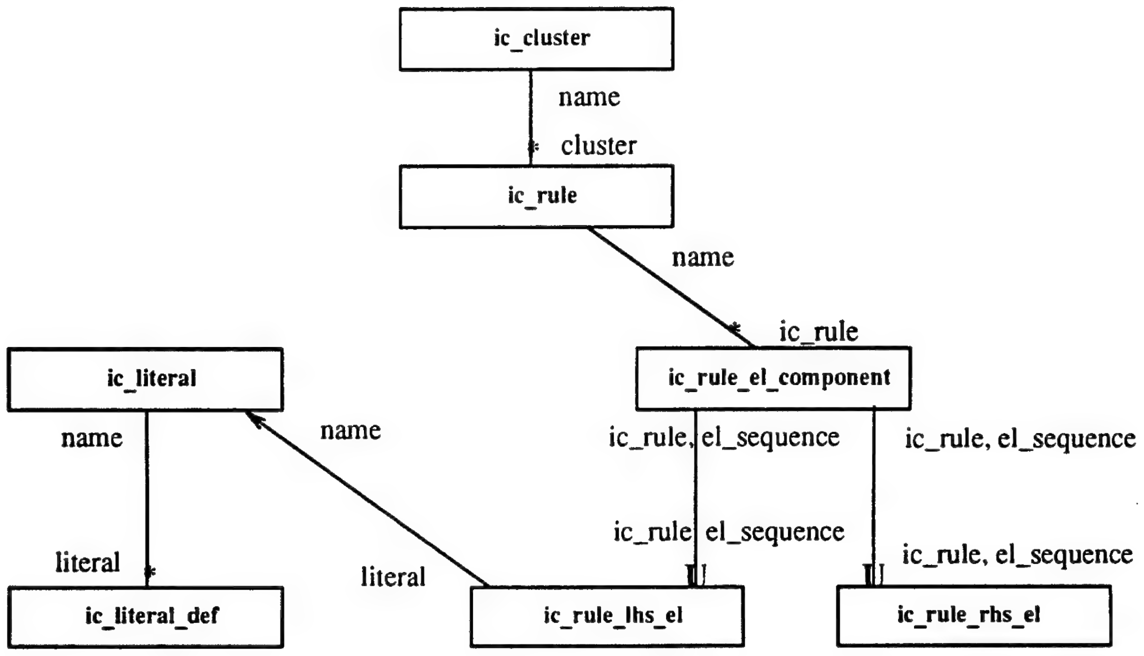
B.1.2 G2: Entities, Attributes, and Their Domains



B.1.3 G3: Tables for Lexicon Definition



B.1.4 G4: Tables for Integrity Constraint Representation



B.2 Alphabetical Listing of Table Definitions

0. **Column Definition Format:**

#	*attribute*	*type*	*note*

1. **dic_noun:** G1

1	singular	varchar (64)	e.g., "(authored) book"
2	plural	varchar (64)	e.g., "(authored) books"
3	countp	char (5)	{"plus" "minus"}

2. **dic_prep:** G1

1	string	varchar (64)	e.g., "since"
2	pred	varchar (8)	e.g., "gt"

3. **dic_verb_form:** G1

1	string	varchar (64)	e.g., "were written by"
2	form	char (6)	e.g., "beppby"
3	num	char (2)	e.g., "pl"

4. **ic_cluster:** G4

1	name	varchar (64)	*a collection of similar rules*
2	note	varchar (255)	

5. **ic_literal:** G4

1	name	varchar (64)	e.g., "teach"
2	attribute_count	smallint	

6. **ic_literal_def:** G4

1	literal	varchar (64)	*– ic_literal.name
2	attribute	varchar (64)	e.g., "instructor"
3	sequence	smallint	*relative position*
4	note	varchar (255)	*explanation string*

7. **ic_rule:** G4

1	name	varchar (64)	e.g., "univ-cs400-constraint"
2	message	varchar (255)	*a message to user*
3	type	char (10)	
4	cluster	varchar (64)	*– ic_cluster.name

8. **ic_rule_el_component:** G4

1	ic_rule	varchar (64)	*– ic_rule.name
2	position	char (1)	{"l" "r"}
3	el_sequence	smallint	*element sequence number*
4	comp_sequence	smallint	*component sequence number*
5	attribute	varchar (64)	→ ic_literal_def.attribute
6	expression	varchar (255)	*value*

9. **ic_rule_lhs_el:** G4

1	ic_rule	varchar (64)	*– ic_rule_el_component.ic_rule
2	el_sequence	smallint	{"l" "r"}
3	el_var	varchar (64)	*an OPS5 element variable*
4	sign	smallint	{"+" "-"}, *postive or negative literal*
5	literal	varchar (64)	→ ic_literal.name

10. **ic_rule_rhs_el:** G4

1	ic_rule	varchar (64)	*– ic_rule_el_component.ic_rule
2	el_sequence	smallint	{"l" "r"}
3	action	varchar (64)	{"make" "modify" "write" ...}
4	obj	varchar (255)	*a qualified OPS5 action object*

11. **kb_attr_domain:** G2

1	domain	varchar (64)	e.g., "pdom.book.title"
2	parent	varchar (64)	→ kb_attr_domain.domain
3	datatype	varchar (64)	e.g., "varchar"
4	ref_db_table	varchar (64)	*reference DB table*
5	ref_db_field	varchar (64)	*reference DB field*

12. **kb_comparator:** G2

1	comparator	varchar (64)	e.g., "gt"
2	arity	smallint	e.g., 2

13. **kb_db_join_edge:** G2, *Implementation of Table 6.2*

1	id	smallint	
2	pred	varchar (8)	
3	table1	varchar (64)	
4	field1	varchar (64)	
5	table2	varchar (64)	
6	field2	varchar (64)	

14. **kb_domain_comparator:** G2

1	domain	varchar (64)	*– kb_attr_domain.domain
2	comparator	varchar (64)	*– kb_comparator.comparator

15. **kb_entity:** G1, G2

1	name	varchar (64)	e.g., "book"
2	parent	varchar (64)	→ kb_entity.name
3	noun	varchar (64)	→ dic_noun.singular
4	active_p	bit	

16. **kb_entity_attr:** G2

1	entity	varchar (64)	*– kb_entity.name
2	attr	varchar (64)	e.g., "title"
3	position	smallint	*relative column position*
4	domain	varchar (64)	e.g., "dom.reference.title"
5	db_table	varchar (64)	*DB table mapping,* e.g., "reference"
6	db_field	varchar (64)	*DB field mapping,* e.g., "title"
7	noun	varchar (64)	e.g., "title"
8	num	char (2)	{"sg" "pl"}
9	prenom_p	bit	*1 if qualified for prenoun modifier*
10	key_attr_p	bit	*1 if part of key*
11	descriptive_p	bit	*1 if default projection attribute*

17. **kb_entity_db_join:** G2

1	entity	varchar (64)	*– kb_entity_attr.entity
2	attr	varchar (64)	*– kb_entity_attr.attr
3	join	smallint	→ kb_db_join_edge.id

18. **kb_entity_derived_attr:** G2

1	entity	varchar (64)	*– kb_entity_attr.entity, e.g. "book"
2	attr	varchar (64)	*– kb_entity_attr.attr, e.g., "keyword"
3	rel	varchar (64)	e.g., "author_book"
4	mod	varchar (64)	e.g., "on_keyword"
5	mod_entity_attr	varchar (64)	e.g., "string"

19. **kb_modifier:** G1

1	name	varchar (64)	e.g., "on_keyword"
2	parent	varchar (64)	→ kb_modifier.name
3	wh_adv	varchar (64)	e.g., "when"
4	entity	varchar (64)	→ kb_entity.name

20. **kb_overlap:** G1

1	entity	varchar (64)	→ kb_entity.name, e.g., "thesis"
2	overlapping_entity	varchar (64)	→ kb_entity.name, e.g., "techreport"

21. **kb_possessive:** G1

1	rel	varchar (64)	e.g., "author_book"
2	possessive_role	varchar (4)	e.g., "obj"
3	cardinality	char (2)	e.g., "pl"

22. **kb_rel:** G1

1	name	varchar (64)	e.g., "author_book"
2	arity	smallint	{1, 2}, e.g., 2
3	parent	varchar (64)	→ kb_rel.name, e.g., "author_reference"
4	tense	varchar (8)	{"past" "pres" "future"}, e.g., "past"

23. **kb_rel_entity_role:** G1

1	rel	varchar (64)	e.g., "author_book"
2	role	varchar (4)	{"subj" "obj"}, e.g., "subj"
3	entity	varchar (64)	e.g., "author"
4	polarity	char (1)	{"+" "-"}, *PSET/NSET membership*

24. **kb_rel_modifier:** G1

1	rel	varchar (64)	*– kb_rel.name
2	modifier	varchar (64)	*– kb_modifier.name

25. **kb_rel_role_conn:** G1, *Implementation of Table 6.3*

1	rel	varchar (64)	*– kb_rel.name
2	pred	varchar (8)	{"=" ">" ...}
3	concept1	varchar (64)	*entity name*
4	table1	varchar (64)	*DB table name*
5	field1	varchar (64)	*DB field name*
6	concept2	varchar (64)	entity or modifier name
7	table2	varchar (64)	*DB table name*
8	field2	varchar (64)	*DB field name*

26. **kb_view:** *Subset views of schema concepts*

1	concept	varchar (64)	*concept names*
2	type	varchar (64)	{"entity" "rel" "modifier"}
3	view_set	varchar (64)	*unique view name*

27. **lex_menu_window:** G3

1	id	varchar (64)	e.g., "concept-noun"
2	title	varchar (64)	e.g., "Concept Noun"
3	width	smallint	*bitmap width of window*
4	position	smallint	*relative position of window*

28. **lex_preterminal:** G3

1	cat	varchar (64)	e.g., "entity-set-n"
2	menu	varchar (64)	→ lex_menu_window.id
3	type	varchar (4)	*domain-dependent or keyword*
4	demon_fn	varchar (64)	*default demon fn to be inherited*
5	before_fn	varchar (64)	*before function*
6	pivot_feature	varchar (64)	

29. **lex_preterminal_feature:** G3

1	cat	varchar (64)	*– lex_preterminal.cat
2	feature	varchar (64)	e.g., "entity"
3	position	smallint	*relative position*
4	unify_fn	varchar (64)	*special unify function*
5	note	varchar (255)	*help string*

30. **lex_prep_modifier:** G1

1	prep	varchar (64)	→ dic_prep.string
2	pred	varchar (8)	→ dic_prep.pred
3	modifier	varchar (64)	*– kb_modifier.name
4	help	varchar (255)	*a help string*

31. **lex_verb:** G1

1	verb	varchar (64)	→ dic_verb_form.string
2	rel	varchar (64)	*– kb_rel.name
3	form	char (6)	→ dic_verb_form.form
4	num	char (2)	→ dic_verb_form.num
5	help	varchar (255)	*a help string*

32. **lex_item:** G3, *Only contains keyword category entries.*

1	id	smallint	*a unique number*
2	cat	varchar (64)	*– lex_preterminal.cat
3	string	varchar (80)	*choice string*
4	help	varchar (255)	*mouse documentation string*
5	demon_fn	varchar (64)	*demon function*

33. **lex_item_feature:** G3, *Only contains keyword category entries.*

1	id	smallint	*– lex_item.id
2	feature	varchar (64)	→ lex_preterminal_feature.feature
3	value	varchar (80)	*a constant*
4	polarity	bit	*1 if part of PSEET; 0 otherwise.*

Bibliography

[1] AICorp, Inc. INTELLECT: Natural language system, conversational English access to corporate databases. Brochure. 138 Technology Drive, Waltham, MA 02154.

[2] Hassan Ait-Kaci and Roger Nasr. LOGIN: A logic programming language with built-in inheritance. *J. Logic Programming*, 3(3):185–215, 1986.

[3] James Allen. *Natural Language Understanding.* The Benjamin/Cummings Publishing Co, 1987.

[4] Madeleine Bates. Rapid porting of the PARLANCE natural language interface. In *Proc. Speech and Natural Language Workshop*, pages 83–88. DARPA/ISTO, February 1989. BBN Systems and Technologies Corp.

[5] Madeleine Bates, M.G. Moser, and David Stallard. The IRUS transportable natural language database interface. In Larry Kerschberg, editor, *Expert Database Systems: Proceedings from the First International Workshop*, pages 617–630. Benjamin/Cummings Publishing Co., 1986.

[6] Robert J. Bobrow and Madeleine Bates. Design dimensions for non-normative understanding systems. In *Proc. 22th Annual Meeting of ACL*, 1982.

[7] Richard R. Burton. *Semantic Grammar: A Technique for Efficient Language Understanding in Limited Domains.* PhD thesis, University of California, Irvine, 1976.

[8] J. G. Carbonell and P. J. Hayes. Recovery strategies for parsing extragrammatical language. Technical Report CMU-CS-84-107, Dept. of Computer Science, CMU, February 1984.

[9] Jaime Carbonell, Barbara Grosz, Wendy Lehnert, Mitchell Marcus, Raymond Perrault, and Robert Wilensky. White paper on natural language processing. In *Proc. Speech and Natural Language*, pages 481–493. DARPA/ISTO, October 1989.

[10] Stuart K. Card. Visual search of computer command menus. In H. Bourma and D. Bouwhuis, editors, *Attention and Performance X*, pages 97–108. Hillsdale, NJ: Erlbaum, 1984.

[11] Stuart K. Card, Thomas P. Moran, and Allen Newell. *The Psychology of Human-Computer Interaction.* Lawrence Erlbaum Associates, 1983.

[12] Sang K. Cha. Kaleidoscope: A cooperative menu-guided query interface (SQL version). In *Proc. IEEE Artificial Intelligence Applications*, pages 304–310, Santa Barbara, California, March 1990.

[13] Sang K. Cha. Kaleidoscope: A cooperative menu-guided query interface (SQL version). *IEEE Trans. on Knowledge and Data Engineering*, 3(1):42–47, March 1991.

[14] Sang K. Cha and Gio Wiederhold. Kaleidoscope data model for an English-like query language. In *Proc. 17-th Conf. on VLDB*, September 1991.

[15] Surajit Chaudhuri. Generalization and a framework for query modification. In *Proc. IEEE Data Engineering Conf.*, Feb 1990.

[16] Peter Chen. The Entity-Relationship Model – toward a unified view of data. *ACM Trans. on Database Systems*, 1(1):9–36, 1976.

[17] Peter Chen. Entity-Relationship diagrams and English sentence structures. In Peter Chen, editor, *Int. Conf. on Entity-Relationship Approach to Systems Analysis and Design.* North-Holland Publishing Company, 1980.

[18] E. F. Codd. A relational model of data for large shared data banks. *Communications of ACM*, 13(6):377–387, 1970.

[19] E. F. Codd. Relational database: A practical foundation for productivity. *Communications of ACM*, 25(2):109–117, Feb 1982.

[20] Thomas A Cooper and Nancy Wogrin. *Rule-based Programming with OPS5.* Morgan Kaufmann, 1988.

[21] Francisco Corella, S. J. Kaplan, G. Wiederhold, and L. Yesil. Cooperative responses to boolean queries. In *Proc. IEEE Data Engineering Conf.*, pages 77–93, April 1984.

[22] Fred J. Damerau. Operating statistics for the transformational question answering system. *Computational Linguistics*, 7(1), 1981.

[23] Knuth Donald E. Semantics of context-free languages. *Mathematical Systems Theory*, 2(2):127–145, 1968. Errata, *Mathematical Systems Theory*, 2(1):95–96 1971.

[24] C. J. Fillmore. The case for case. In E. Bach and R. Harms, editors, *Universals in Linguistic Theory.* Holt, Rinehart, and Winston, 1968.

[25] C. L. Forgy. OPS5 user's manual. Technical Report CMU-CS-81-135, Carnegie-Mellon University, Pittsburgh, PA, 1981.

[26] Annie Gal and Jack Minker. Informative and cooperative answers in databases using integrity constraints. Technical Report CS-TR-1191, University of Maryland, September 1987.

[27] Jay Glicksman, Jeff Y.-C. Pan, Jay M. Tenenbaum, and Bruce L. Hitson. A central knowledge service for a distributed cim environment. Schlumberger Technologies and Center for Integrated Systems, Stanford University, 1990.

[28] H. P. Grice. Logic and Conversation. In Donald Davidson and Gilbert Harman, editors, *The Logic of Grammar*, pages 64–75. Dickinson Publishing Co, 1975.

[29] Barbara Grosz, Douglas E. Appelt, Paul Martin, and Fernando Pereira. TEAM: An experiment in the design of transportable natural language interfaces. Technical Report 356, SRI AI center, Aug 1985.

[30] Charles T. Hemphill, Inderjeet Mani, and Steven L. Bossie. Towards an effective natural language interface to knowledge based systems. Internal Working Paper, AI lab., Computer Science Center, Texas Instruments, Inc., Dallas, TX, December 1986.

[31] Gary G. Hendrix, Earl D. Sacerdoti, Daniel Sagalowicz, and Jonathan Slocum. Developing a natural language interface to complex data. *ACM Trans. on Database Systems*, 3(2):105–147, June 1978.

[32] Jurgen M. Janas. On the feasibility of informative answers. In H. Gallaire, J. Minker, and J. M. Nicolas, editors, *Advances in Database Theory*, pages 397–414. Plenum Press, 1981.

[33] Matthias Jarke, Jon A. Turner, Edward A Stohr, Yannis Vassiliou, Norman H. White, and Ken Michielsen. A field evaluation of natural language for data retrieval. *IEEE Trans. Software Engineering*, SE-11(1):97–114, January 1985.

[34] Mimi Kao, Nick Cercone, and Wo-Shun Luk. Providing quality responses with natural language interfaces: The null value problem. *IEEE Trans. on Software Engineering*, 14(7):959–984, July 1988.

[35] S. Jerrold Kaplan. Cooperative responses from a portable natural language query system. *Artificial Intelligence*, 19:165–187, October 1982.

[36] M. Kay. Algorithm schemata and data structures in syntactic processing. In Barbara J. Grosz, Karen Sparck Jones, and Bonnie Lynn Webber, editors, *Readings in Natural Language Processing.* Morgan Kaufmann Publishers, Inc, 1986.

[37] H. J. Kim, H. F. Korth, and A. Silberschatz. PICASSO: A graphical query language for naive end users. Technical Report TR-85-30, Dept. of Computer Science, Univ. of Texas at Austin, November 1985.

[38] Jonathan J. King. *Query Optimization by Semantic Reasoning.* PhD thesis, Stanford University, May 1981. Also published by University of Michigan Press, 1984.

[39] Kevin Knight. Unification: A multidisciplinary survey. *ACM Computing Surveys*, 21(1):93–124, March 1989.

[40] Michel Kunz and Rainer Melchart. Pasta-3's graphical query language: Direct manipulation, cooperative queries, full expressive power. In *Proc. 15th Conf. on VLDB*, 1989.

[41] Mitchell P. Marcus. Building non-normative systems – the search for robustness. In *Proc. 20th Annual Meeting of ACL*, 1982.

[42] Eric Mays. Failures in natural language systems: Applications to data base query systems. In *Proc. AAAI*, pages 327–330, 1980.

[43] Kathleen Filliben McCoy. *Correcting Object-Related Misconceptions.* PhD thesis, University of Pennsylvania, Dec 1985.

[44] G. A. Miller. The magical number seven, plus or minus two: Some limits on our capacity for processing information. *The Psychological Review*, 63(2):81–97, March 1956.

[45] Jack Minker, editor. *Foundations of Deductive Databases and Logic Porgramming.* Morgan Kaufmann Publishers, Inc., San Mateo, CA, 1988.

[46] Amihai Motro. Query generalization: A method for interpreting null answers. In Larry Kerschberg, editor, *Expert Database Systems: Proceedings from the First International Workshop*, pages 597–616. Benjamin/Cummings, 1986.

[47] Natural Language, Inc. How to choose a natural language interface for your database. Brochure, 1989. 2910 Seventh Street, Berkeley, CA 94710, U.S.A.

[48] Donald A. Norman. Design principles for human-computer interfaces. In Ronald M. Baecker and William A.S. Buxton, editors, *Readings in Human-Computer Interaction*, pages 492–501. Morgan Kaufmann, 1987. Also in Proc. of ACM CHI'83.

[49] Donald A. Norman. *The Psychology of Everyday Things.* Basic Books, Inc., 1988.

[50] Jeff Y-C Pan and Jay M. Tenenbaum. An intelligent agent framework for enterprise integration. Schlumberger Technologies and Center for Integrated Systems, Stanford University, 1990.

[51] Gary Perlman. Making the right choices with menus. In *Human-Computer Interaction — Interact' 84, Amsterdam*, pages 317–321. North Holland, Amsterdam, 1985.

[52] Brian Phillips, Michale J. Freiling, James H. Alexander, Steven L. Messick, Steve Rehfuss, and Sheldon Nicholl. An eclectic approach to building natural language interfaces. In *Proc. 26th Annual Meeting of ACL*, 1986.

[53] Brian Phillips and Sheldon Nicholl. INGLISH: A natural language interface. In *Foundation for Human-Computer Communication.* IFIP WG 2.6 Working Conference on The Future of Command Languages, 1985.

[54] Xioalei Qian. *The Deductive Synthesis of Database Transactions.* PhD thesis, Stanford University, November 1989. Also as Technical Report No. STAN-CS-89-1291.

[55] Phyllis Reisner. Use of psychological experimentation as an aid to development of a query language. *IEEE Transactions on Software Engineering*, SE-3(3), 1977.

[56] Phyllis Reisner. Human factors studies of database query languages: A survey and assessment. *ACM Computing Surveys*, 13(1), 1981.

[57] Phyllis Reisner, Raymond F. Boyce, and Donald D. Chamberlin. Human factors evaluation of two data base query languages - SQUARE and SEQUEL. In *Proc. National Computer Conf.* AFIP Press, 1975.

[58] Roger C. Schank and Charles J. Rieger III. Inference and the computer understanding of natural language. *Artificial Intelligence*, 5:373–412, 1974.

[59] Stuart M. Shieber. *An introduction to Unification-Based Approaches to Grammar.* Center for the Study of Language and Information, Stanford University, 1985.

[60] Ben Shneiderman. *Software Psychology: Human factors in Computer and Information Systems.* Winthrop Publishers, Inc., 1980.

[61] Robert F. Simmons. Man-machine interface: Can they guess what you want? *IEEE EXPERT*, pages 86–93, 1986.

[62] Herbert A. Simon. *The Sciences of the Artificial.* MIT Press, 2 edition, 1981.

[63] M. Stonebraker, M. Hearst, and S. Postamianos. A commentary on the Postgress rule system. *ACM SIGMOD Record*, 18(3):5–11, September 1989.

[64] Sybase. *TRANSACT-SQL manual*, 1988.

[65] Marjorie Templeton. Problems in natural-lanugage interface to DBMS with examples from EUFID. In *Proc. 23rd Annual Meeting of ACL*, 1983.

[66] Marjorie Templeton and John Burger. Considerations for development of natural-language interfaces to database management systems. In L. Bolc and M. Jarke, editors, *Cooperative Interfaces to Information Systems*, chapter 3, pages 67–99. Springer-Verlag, 1986.

[67] Harry Tennant. The commercial application of natural language interfaces. A preprint for COLING 86 Panel Discussion *Natural Language Interfaces – Ready for Commercial Success?*, 1986.

[68] C. Thompson, S. Corey, M. Rajinikanth, P. Bose, S. Martin, R. Roberts, R. Lewis, R. Enand, T. DiPesa, and S. Cha. RTMS: Toward close integration between database and application. In *Proc. of 20-th Annual Hawaii Int'l Conf. on System Sciences*, 1987.

[69] C. W. Thompson, K. M. Roth, H. R. Tennant, and R. M. Saenz. Building usable menu-based natural language interface to databases. In *Proc. 9th Conf. on VLDB*, pages 43–55, 1983.

[70] Craig W. Thompson. Beyond Retrieval: Updating a Database using Menu-Based natural Language Understanding. In *Conference on Intelligent Systems and Machines*, April 1984.

[71] Craig W. Thompson. *Using Menu-based Natural Language Understanding to Avoid Problems Associated with Traditional Natural Language Interfaces to Databases.* PhD thesis, Univ. of Texas, Austin, May 1984. Also published as CSL Tech Report 84-12, Texas Instruments, Inc.

[72] David L. Waltz, Timothy Finin, Fred Green, Forrest Conrad, Bradley Goodman, and George Hadden. The planes system: Natural language access to a large data base. Technical Report T-34, Coordinated Science Lab., Univ. of Illinois, Nov 1976.

[73] Charles Welty and David W. Stemple. Human factors comparison of a procedural and nonprocedural query language. *ACM Trans. on Database Systems*, 6(4), 1981.

[74] Jennifer Widom and Sheldon J. Finkelstein. A syntax and semantics for set-oriented production rules in relational database systems. Technical report, IBM Almaden Research Center, 1989.

[75] Gio Wiederhold. *Database Design.* McGraw-Hill, second edition, 1983.

[76] Gio Wiederhold. Views, objects, and databases. *IEEE Computer*, 19(12):37–44, December 1986.

[77] Terry Winograd. *Language as a Cognitive Process : Syntax.* Addison Wesley, 1983.

[78] W. A. Woods. Transition network grammars for natural language analysis. *Comm. of ACM*, 3(10):591–606, 1970.

[79] Stanley B. Zdonik and David Maier, editors. *Reading in Object-oriented Database Systems.* Morgan Kaufmann Publishers, Inc., San Mateo, CA, 1990.